2004

# Architectural level risk assessment

Ahmed E. Hassan
*West Virginia University*

# Architectural Level Risk Assessment

**Ahmed E. Hassan**

Dissertation submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Engineering

Hany H. Ammar, Ph.D., Chair
Katerina D. Goseva-Popstojanova, Ph.D., Co-Chair
Ali Feliachi, Ph.D.
Bhaskaran Gopalakrishnan, Ph.D.
Sherif Yacoub, Ph.D.

Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2004

Keywords: Risk Assessment, Severity Analysis, UML, Software Metrics

## ABSTRACT

### Architectural Level Risk Assessment

Ahmed E. Hassan

Many companies develop and maintain different types of large-scale software systems for public and financial institutions. Should a failure occur in one of these systems, the impact would be enormous. It is therefore essential, in maintaining a system's quality, to identify any defects early on in the development process in order to prevent the occurrence of failures. However, testing all modules of large-scale systems to identify defects can be very expensive. There is therefore a need for methodologies and tools that support software engineers in identifying the defected and complex software components early on in the development process. Accurate early estimates will help reduce wasted resources associated with testing.

Risk assessment is an essential process for ensuring high quality software products. By performing risk assessment during the early software development phases we can identify complex and high risk software modules, thus enables us to enhance resource allocation decisions.

To assess the risk of software systems early on in the software's life cycle, we propose an architectural level risk assessment methodology. It combines the probability of software failures and the severity associated with these failures to estimate software risk factors. As a result, remedial actions to control and improve the quality of the

software product can be taken. We use UML specifications of software systems, which are available early on in the software life cycle to come up with the risk factors of software architectural elements (components/connectors), the scenarios, the use cases and systems. Based on this methodology we build a risk assessment model which will enable us to identify complex and noncomplex software components. We will be able to estimate programming and service effort, and estimate testing effort. This model will enable us also to identify components with high risk factor which would require the development of effective fault tolerant mechanisms.

To estimate the probability of software failure we introduced and developed dynamic metrics which are used to measure dynamic complexity and dynamic coupling for software architectural elements from UML static models.

To estimate severity of software failure we propose an architectural level severity methodology early on in the software design phase based on UML artifacts. Also we propose a validation process for both risk and severity methodologies. Finally we propose to implement a prototype tool support for the automation of the risk assessment methodology.

# ACKNOWLEDGMENTS

First, thanks to God the most merciful. Second, I would like to express my deepest gratitude and appreciation to my research advisor, Dr. Hany Ammar, for this opportunity he gave me to conduct research under his supervision, for helping me define my research goals, for his support and review and for valuable guidance during this research. I would like to thank Dr. Katerina Goseva-Popstojanova for serving as a co-chair, for her ever presence guidance during this research effort and I thank her very much for her motivation and guidance throughout my research program. I would also like to thank my committee members, Dr. Bhaskaran Gopalakrishnan for his valuable time he gave me, Dr. Ali Feliachi for his valuable advice and for the valuable time he gave me to serve as member of my graduate committee and Dr. Sherif Yacoub who has laid the ground to this research. I am also grateful to all my colleagues in the research lab. Thank you for your help and for creating a positive atmosphere that constantly motivated to expand my limits. I would like also to thank my brother, my sisters and my wife for help and support they gave me.

I gratefully acknowledge the financial support for my research provided by the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) managed through the NASA Independent Verification and Validation (IV&V) Facility in Fairmont, West Virginia.

TABLE OF CONTENTS

LIST OF TABLES

## LIST OF FIGURES

## LIST OF ACRONYMS/ABBREVIATIONS

| | |
|---|---|
| UML | Unified Modeling Language. |
| OO | Object Oriented |
| MPC | Message Passing Coupling |
| HAZOP | Hazard and Operability Studies |
| FMEA | Failure Mode and Effects Analysis |
| FMECA | Effects and Criticality Analysis |
| FTA | Fault Tree Analysis |
| FHA | Functional Hazard Assessment |
| ETA | Event Tree Analysis |
| FFA | Functional Failure Analysis |
| COTS | Commercial Off-The-Shelf software |
| $S_x$ | Scenario x in a UML model |
| $CC$ | Component Cyclometric complexity |
| $e$ | The number of edges in the control flow graph |
| $n$ | The number of nodes in the control flow graph. |
| $C_i^x$ | The subset of states for a component $i^{th}$ visited in the scenario $S_x$ |
| $T_i^x$ | The subset of transitions traversed in the state chart of component $i^{th}$ in the scenario $S_x$ |
| $c_i^x$ | The number of nodes in this graph is; (the cardinality of $C_i^x$) |

| | |
|---|---|
| $t_i^x$ | The number of edges in this graph is; (the cardinality of $T_i^x$) |
| $dco_i^x$ | The dynamic complexity of component $i^{th}$ in scenario $S_x$ |
| $DOC_i^x$ | The normalized dynamic complexity of $i^{th}$ component in scenario $S_x$ |
| $EOC_{ij}^x$ | Normalized dynamic coupling of a connectors between $i^{th}$ and $j^{th}$ components in scenario $S_x$ |
| $MT_{ij}^x$ | The set of messages sent from component $i^{th}$ to component $j^{th}$ during the execution of scenario $S_x$ |
| $MT^x$ | The set of all messages exchanged between all components active during the execution of scenario $S_x$ |
| *CD* | *CoilDrive r* |
| *CG* | *CommunicationGnome* |
| *AR* | *AtrialModel* |
| *VT* | *VentricularModel* |
| *AVI* | *A*trial *V*entricular *I*nhibited "operational mode" |
| *AAI* | *A*trial *A*trial *I*nhibited "operational mode" |
| *AAT* | *A*trial *A*trial *T*riggered "operational mode" |
| *VVI* | *V*entricular *V*entricular Inhibited "operational mode" |
| *VVT* | *V*entricular *V*entricular *T*riggered "operational mode" |
| S | Software system |
| *Uc* | Use Case |

| | |
|---|---|
| *Act* | Use case actor |
| C | Software component |
| *E* | Events (message exchange between scenario and environment) |
| ${}^{x}Cost_i(M)$ | The cost of failure of (component/connector) $i^{th}$ in a given failure mode $M$ in a given scenario $S_x$ |
| ${}^{x}p_i(M)$ | The probability of (component/connector) $i^{th}$ being in failure mode $M$ in a given scenario $S_x$ |
| ${}^{x}p(H)$ | The probability of system level hazard $H$ for a given scenario $S_x$ |
| ${}^{x}Cost(H)$ | Cost of failure for a given system hazard $H$ in a scenario $S_x$ |
| ${}^{x}Cost(H)$ | Cost of failure for a given system hazard $H$ in a scenario $S_x$ |
| ${}^{x}TotalCost_i$ | Total expected cost of failure of (component/connector) $i^{th}$ in a given system scenario $S_x$ |
| ${}^{x}p(S)$ | Probability of execution of a given scenario $S_x$ |
| ${}^{x}TotalCost(S)$ | The total expected cost of failure of a given scenario $S_x$ |
| $rf_i^x$ | The risk factor of a component $i^{th}$ in scenario $S_x$ |
| $svt_i^x$ | The severity level for the $i^{th}$ component in the scenario $S_x$ |
| $rf_{ij}^x$ | The risk factor for a connector between components $i^{th}$ and $j^{th}$ in the scenario $S_x$ |

| | |
|---|---|
| $svt_{ij}^{x}$ | The severity level for the connector between the $i^{th}$ and the $j^{th}$ components in the scenario $S_x$ |
| DTMC | Discrete Time Markov Chain |
| $P_{ij}^{x}$ | The conditional probability that the program will next execute component $j^{th}$, given that it has just completed the execution of the component $i^{th}$ |
| $P_k^x$ | The probability of occurrence of scenario $S_x$ in use case *k* |
| $P_k$ | The probability of occurrence of use case *k* |
| $P_{AVI}$ | The transition probability matrix for the *AVI* scenario |
| CO | COmpleteness measure (Quality achieved) of the predictive model |
| $n_{11}$ | The number of components which have low risk factor and it is predicted as having low risk factor in software system S |
| $n_{12}$ | The number of components which have low risk factor and it is predicted as having high risk factor in software system S |
| $n_{21}$ | The number of components which have high risk factor and it is predicted as having low risk factor in software system S |
| $n_{22}$ | The number of components which have high risk factor and it is predicted as having high risk factor in software system S |
| $N_{1+}$ | Number of low risk factor components in the software system *S* |
| $N_{2+}$ | Number of high risk factor components in the software system *S* |

| | |
|---|---|
| $N_{+1}$ | The number of components predicted as Low risk in the software system *S* |
| $N_{+2}$ | The number of components predicted as High risk in the software system *S* |
| $N$ | The number of all components in the software system *S* |
| $A$ | Correctness value (Proportion correct) of the predictive model |
| $f$ | The specificity of the predictive model |
| $S$ | The sensitivity of the predictive model |
| CCS | Command and Control System |
| CSCI | Computer Software Configuration Item |
| RTS | Remote Transmission System |
| ARAT | Architecture-level Risk Assessment Tool |
| RRT | Rational Rose Real Time |

# *Chapter 1*

## Introduction and Background

This section introduces the research motivation and presents an overview of the general problem area. It describes the significance of the research and gives an overview of the research including the scope and limitations. Background information follows to establish fundamental concepts.

### 1.1 Motivation

All software projects are exposed to some degree of risk. The process of developing a software solution based on plans and schedules containing estimates, assumptions and other uncertainties are a risky business. This risk arises from software project [Heemstra, 2003], software process [Brockers, 1995] or software product [Katerina, 2003]. The software product risk is the risk of failure of software product. Our research focus in this dissertation is software product risk assessment.

Many critical systems rely for their correct operation on complicated software systems [Knight, 2000]. The impact of failure of these systems is huge. It is therefore very important to identify any defects in these software systems in advance to reduce the occurrence of failures. However, testing all modules in these types of large-scale systems to identify defects is very expensive.

Accurate early estimates can help reduce wasted resources associated with testing. Elemam et al [Elemam, 1999f] suggests that most field faults in software are found in a

small proportion of the software components. Also Fenton et al [Fenton, 1999f] found very strong indications that a small number of software components contain most of the faults discovered during the testing phase, and that a very small number of components contain most of the faults discovered in operation. This means that if these faulty components can be detected early on in the software life cycle development, rectifying actions can be taken. If we can identify complex and critical software components in advance, we can focus on the components that need intensive testing and hence detect defects more efficiently.

Risk assessment is an essential process in managing and controlling software development process. Risk assessment at the early design phase of the software is more feasible and more beneficial than assessment at later development phases. Architecture level risk assessment can be used to guide software development, testing, and maintenance process.

The risk assessment at the software design phase will allow for a wide range of preventive/corrective actions to be taken with the least impact on budget and schedule. It enables designers to backtrack and redesign components with high risk factor. It could also be used for assigning technical staff to testing, and maintenance efforts. Rather than inspecting all software components on an equal basis, focusing on high risk components can improve the efficiency of inspection.

To estimate risk of a software system we have to find an estimate for the probability of software failure and also an estimate for the severity of this failure [NASA3, 2000]. Our work is motivated by systemizing the risk assessment process. In doing so, we estimate the probability of software failure as a function of software dynamic behavior and propose a systematic process for the severity of software failure.

## 1.2 Research Significance

We have developed a risk assessment methodology to assess the risk of software systems based on measurable parameters that can be automatically collected and analyzed in the early software design phase based on UML artifacts.

In summary our contributions include:

- Introducing an architectural level risk assessment methodology.
- Introducing and developing UML based dynamic metrics which can be used as a measure for probability of software failure.
- Introducing and developing an architectural level severity assessment technique using classical hazard analysis methods and software UML models.
- Developing a tool support for the automation of the risk methodology.

## 1.3 Dissertation Structure

Chapter 2 presents the research objective of this work. Chapter 3 contains a literary survey of related work in areas of software metrics, hazard analysis, severity analysis and risk assessment. Chapter 4 addresses the dynamic metrics based on UML static models. Chapter 5 presents the risk assessment methodology and risk model for component/connector, scenario and use case risk model. Chapter 6 describes the severity assessment methodology, Chapter 7 presents case studies, Chapter 8 is the evaluation criteria for the risk assessment model, Chapter 9 presents a prototype tool support, and. conclusions and future work are presented in Chapter 10.

## 1.4 Background

This section contains relevant background information to orient the reader, provide a foundation of basic risk concepts, clarify the meaning of software risk assessment, and define key terms used in this dissertation.

### 1.4.1 Risk assessment

NASA-STD-8719.13A standard [NASA2, 1997] defines risk as a combination of two factors: probability of a malfunction (failure) and the consequence of that malfunction (severity). Probability of failure depends on the probability of occurrence of a fault combined with the likelihood of exercising that fault. This standard defines several types of risks, for example, availability risk, acceptance risk, performance risk, cost risk, schedule risk, and reliability based risk. Our interest is the reliability-based risk. The reliability-based risk takes into account the probability that the software product will fail in the operational environment and the adversity of that failure [Yacoub, 2003].

### 1.4.2 Unified Modeling Language (UML)

The Unified Modeling Language (UML) [OMG, 2000] is a widely accepted standard notation for modeling software systems and its use is continuously growing. The software development industry is embracing UML language for its various uses, starting from requirement analysis, to define software system architecture and also in the subsequent phases of software life cycle. UML provides a framework for decomposing the problem of software design into smaller components that are related to one another. Different UML diagrams are provided (in an integrated framework) to represent the software model from different viewpoints. The UML language is supported by graphical

representations (easy to use), that are not far from the classical diagrams used before introducing UML (e.g., state diagrams, class diagrams, and sequence diagrams.

Software architecture is defined in terms of components and connectors in UML models.

We will use the generic term component to refer to the unit of observation. This may mean a procedure, a file, an object, a method and as elaborate as a package of classes or procedures. The proposed methodology is applicable irrespective of the exact definition of a component. Connectors can be as simple as procedure calls; they can also be as elaborate as client-server protocols, links between distributed databases, or middleware. The components are mapped to the various components of the UML sequence diagrams and the connectors can be perceived as the medium through which the message transfer takes place

Next section sheds lights on dynamic metrics captured from UML static specifications.

### 1.4.3 Dynamic Metrics

Static analysis [Benlarbi, 1999] helps software designers in generating software metrics such as class size, the size of the hierarchy and static complexity measures which could help in estimating code level defects. The complex dynamic behavior of many applications, especially real-time applications, motivates a shift in interest from traditional static analysis to dynamic analysis. Dynamic analysis is performed to analyze the behavior of objects as expected during run time. UML defines modeling specifications that can be used to specify the dynamic aspects of the software architecture, which is critical to all development phases. UML is a suitable candidate for

the proposed dynamic metrics. These dynamic metrics could automatically capture the dynamic (complexity, coupling) of software architecture element components/connectors from the UML visual static model.

The value of software metrics stems from their association with measures of important external attributes [Elemam, 1999f]. An external attribute is measured with respect to how the product relates to its environment. Examples of external attributes are testability, reliability and maintainability. Practitioners, whether they are developers, managers, or quality assurance personnel, are concerned with the external attributes. However, they cannot measure many of the external attributes directly until quite late in the life cycle of a project or even a product itself. Therefore, they can use product metrics as leading indicators of those external attributes that are important to them. For instance, if we know that a certain coupling metric is a good leading indicator of maintainability, as measured in terms of the effort to make a corrective change, then we can minimize coupling during design because we know that in doing so, we are also increasing maintainability. As explored in chapter 4 we propose dynamic metrics [Hassan, 2001] based on UML for measuring dynamic complexity/coupling of software architectural elements (i.e. component/connector). We relate these metrics to the probability of component/connector failure [Yacoub, 1999].

### 1.4.4 Probability of Failure

As stated in section 1.4.1, risk is defined as combination of probability of a malfunction (failure) and the consequence of that malfunction (severity).

During the early phases of the software life cycle, it is difficult to estimate the probability of failure of software components; therefore we use quantitative factors such

as complexity (for components) and coupling (for connectors), which have a major impact on fault proneness according to [Elemam11, 1999]. We use dynamic metrics to estimate the probability of fault manifesting into a failure. Dynamic metrics are used to measure the dynamic behavior of the system in a given scenario based on the premise that the active components/connectors are the source of failures [Yacoub3, 1999]. We use dynamic metrics as indicators of the probability of failure of a software component/connector.

### 1.4.5 Severity

Traditional software fault detection models do not take into account the fact that the consequences of various software failures caused by faults can be very different [Briand, 1993a]. Thus, they are of limited use in the allocation of resources to the portions of a system with the greatest risk. Since software failures have different consequences, any measure of software fault proneness must include the measurement of the consequence of failure (severity) [El-Emam, 1999]. Traditional software fault detection models have not considered the cost of failure [Lanubile, 1997], [Harrison, 1988], so they are inappropriate for measuring the true risk associated with failure. Since software failures have different consequences, [Susan, 1988] any measure of software reliability risk must include the measurement of the consequence of failure.

### 1.4.6 Hazard Analysis

Hazard analysis plays a key role in system safety approach. A hazard is “any real or potential condition that can cause injury, illness, or death to personnel, or damage to, or loss of, equipment or property, or damage to environment” [DoD, 1997]. Many techniques are available to help identify and analyze hazards. The use of multiple hazard

analysis techniques is recommended because each has its own purpose, strengths and weaknesses. Typically, each technique addresses certain aspects of safety; thus, one technique alone is not sufficient to identify and analyze all hazards of a system [Sammarco, 2003].

### 1.5 Definitions

***Hazard*****:** *Existing or potential condition that can result in or contribute to, a mishap* [NASA2, 1997].

***Mishap*****:** An unplanned event or series of events that results in death, injury, occupational illness, or damage to or loss of equipment, property, or damage to the environment; an accident [NASA2, 1997].

***Risk*****:** As it applies to safety, exposure to the chance of injury or loss. It is a function of the possible frequency of occurrence of the undesired event, of the potential severity of resulting consequences, and of the uncertainties associated with the frequency and severity [NASA2, 1997].

***Software error*****:** An incorrect step, process, or data definition; for example, an incorrect instruction in a computer program.

***Software failure*****:** An event in which a system or system component does not perform a required function within specified limits [Sammarco, 2004].

***Software fault:*** A manifestation of an error in the software. If encountered, a failure might result [Sammarco, 2004].

***Software architecture*****:** There is no standard, universally-accepted definition of the term, "software architecture," although there is no shortage of definitions, either [Bass, 2003]. The software architecture of a program or computing system is the structure or structures

of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

***Severity*:** severity considers the worst potential consequence of a failure, determined by the degree of injury, property damage, or system damage that could ultimately occur [MIL_STD, 1984].

***Software metrics:*** metrics are means of measuring some aspect of software and, in most organizations, it directly relates to quality [Zuse, 1998]. The IEEE definition is "**metric**" is synonymous with a "**software quality metric**" and it defines a software quality metric as a function with input and output. Software quality metrics have software data as inputs and a single numeric value as output. The output is interpreted as the degree to which software possesses a given attribute that affects its quality [IEEE, 1993].

***Software process:*** Lonchamp [Lonchamp, 1993] defines a software process as "*A set of partially ordered process steps, with a set of related artifacts, human and computerized resources, organizational structures and constraints, intended to produce and maintain the requested software deliverables".*

### 1.6 Summary

We presented in this chapter an overview of the general problem area. It describes the research significance and gives an overview of the research scope and the limitations. Finally, relevant background information has been presented to provide a foundation of basic risk assessment concepts, clarify commonly misunderstood relationships, and introduce terminology, in order to provide a common understanding.

# *Chapter 2*

## Research Objectives and Specific Aims

### 2.1 Overview

In order to improve and control the quality of the software during the software development process, software engineers and managers need methodologies and tools to support software design and decision-making. Early risk assessment based on UML models is a helpful tool for managers as well as software engineers. It could be a very effective method of improving the system quality and reducing testing costs.

Several methodologies for risk assessment have been developed, mostly based on subjective views. These approaches of risk assessment are highly sensitive to the manager's perceptions and preferences, which are difficult to represent by an algorithm. Depending on the decision-maker's attitude towards risk, he or she can decide early with little information, or can postpone the decision, thus gaining time to obtain more information, but losing some control. Without a systematic way of assessing possible risk, it is difficult to build a high quality cost-effective system [Boehm, 1997]. There is a need for a methodology to transform risk assessment into a structured problem with systematic solutions. Constructing a model to assess risk based on objectively measurable parameters that can be automatically collected and analyzed in the early design phase is our focus in this dissertation.

The traditional software fault detection models do not consider the fact that the consequences of various software failures caused by faults will be very different [Briand, 1993a]. Thus, they are of limited use in the allocation of resources to the portions of a

system with the greatest risk. Any measure of software risk must include the measurement of the consequence of failure (severity) [El-Emam, 1999] [Susan, 1988]. We propose a methodology for estimating the consequence of software failure (severity) early on in the software design phase based on UML artifacts.

### 2.2 Research Objective

Our objective is to develop risk assessment methodology based on measurable parameters that could automatically be collected and analyzed in the early software design phase based on UML artifacts. We have also developed a severity assessment methodology based on UML models.

### 2.3 Specific Aims

- Introducing architectural level risk assessment methodology, we should be able to transform risk assessment into a structured problem with systematic solutions.
- Introducing and developing dynamic metrics which are used to measure dynamic complexity and dynamic coupling for software architectural element (component/connector) from UML static models.
- Introducing and developing architectural level severity assessment technique based on UML models using classical hazard analysis methods.
- Building risk assessment model which will enable us to :
    - Identify complex software components that need detailed inspection.
    - Identify noncomplex software components that are likely to have a low risk factor and therefore candidates for development without detailed inspection.
    - Estimate programming and service effort, and estimate testing effort.

- Identify components with high risk factor which would require the development of effective fault tolerance mechanisms.

- Implementing a tool support for the automation of the proposed risk assessment methodology.
- Validating the methodology. For validating the methodology, we will compare our results of the dynamic metrics and the risk factor to the results obtained from the simulation model developed in [Yacoub, 2002].

# *Chapter 3*

## Related Work

This chapter discusses previous related work in software metrics, severity analysis and risk assessment, critiques the limitations and inadequacies of this research. It draws distinctions between prior work and the proposed research. We proceed as follows:

Software metrics review, severity analysis review and software risk assessment review.

### 3.1 Software Metrics Review

Software metrics offer a means to understand the process and product of the software [Zuse, 1998]. It is necessary for software quality control [Fenton, 1999b]. The most significance benefit of software metrics is that they are designed to provide information to support managerial decision making during the software lifecycle. In his research, Shyam et al [Shyam, 1998] recommended that the use of various well-constructed metrics could be a basis for software managerial decision-making and could provide insight into the software design process. Fenton et al [Fenton, 1996] classify software metrics into three classes; resource, process and product metrics. Process and product metrics can help both managing activities, (such as costing, scheduling, controlling and staffing), and engineering activities (such as analyzing, designing, coding, testing and documenting) [Abreu, 1996].

For the purpose of this dissertation our focus is on software product metrics; resource and process metrics are out of the scope of this work.

### 3.1.1 Software Product Metrics

The field of software product metrics has been a major focus of software metrics research [Mills, 1998]. Measuring software products could help in establishing control over software development activities [Littlefair, 2001]. Software metrics research has revealed that software product metrics can be the basis for software quality predictions, such as whether a software module is fault-prone [Taghi, 1999]. Research has shown software product metrics to be useful predictors of software faults [Oman, 1997]. Software product metrics can quantify the size and complexity of software artifacts in many dimensions. It is currently a major determinant of the cost and effort required to develop software product [Mills, 1998]. Product metrics incorporated into software predictive models could give advanced warning of potential risks [Fenton, 1999mm]. Software product metrics could be used to measure software products such as source code or design documents. Because it is simple and easy to automate, the number of lines of source code metrics is one of the earliest software product metrics [Fenton, 1999]. Other examples of product metrics include Halstead [Halstead] and McCabe's Cyclomatic Complexity [McCabe, 1976]. Since the early days of computer science, these traditional metrics are still in use. These traditional metrics could be used to quantify the internal structure of procedural software systems [Zuse, 1991]. However, the need to quantify the unique features of emerged Object Oriented paradigm has given birth, in recent years, to new metric sets. Concepts of Object Oriented (OO) design and development have become popular in today's software development environment and this motivates the shift of interest from traditional software metrics to OO metrics [Chidamber, 1994]. Object oriented development has proved its value for those software systems that must be maintained and modified [Rosenberg, 1998].

The idea behind the development of object oriented metrics is that they can offer early predictors of software components that contain faults (found during testing or that result in field failures) or that are costly to maintain [Benlarbi, 1999] [Briand, 1998]. This could also let the organization to take mitigating actions early and consequently avoiding costly rework.

### 3.1.2 Object Oriented Metrics

A considerable number of object-oriented metrics have been constructed in the past; for example, Abreu et al proposed a set of metrics suitable for evaluating the use of main abstractions of the Object Oriented paradigm such as inheritance, encapsulation, information hiding [Abreu, 1994]. Benlarbi et al [Benlarbi, 1999] proposed a set of OO measures to define and examine the quality impact of polymorphism on OO design. They concluded that these measures are significant predictors of fault proneness. Briand et al proposed a suite of measures to quantify class coupling during the design of object oriented systems [Briand, 1997]. Their results show that some of these coupling measures may be useful as quality indicators of the design of OO systems. Chidamber et al developed a new suite of metrics for OO design [Chidamber, 1994]. Cartwright and Shepperd [Cartwright, 2000] described an empirical investigation into an object oriented (OO) system comprised of 133,000 lines of C++. He proposed a prediction system for size and number of defects based upon OO classes. To achieve more reliable testing Mei et al [Mei, 1999] proposed a set of object oriented metrics that help in selecting effective testing techniques. Kamiva et al [Kamiva, 1999] proposed a new method to estimate the fault proneness of the class in the early design phase, using several complexity metrics for OO software. In this proposed method, they introduced four checkpoints into the analysis / design / implementation phase,

and estimated the fault prone classes using the applicable metrics at each checkpoint. Letha et al [Letha, 2000] introduced a suite of OO metrics, which offer a more direct mapping from the metric to its associated quality factor. Much research has also been done to assess the design quality of OO software systems based on OO metrics [Hitz, 1996], [Briand, 1999b], [Simon, 1997]. Most of these studies did not focus exclusively on those metrics that can be collected during the design stage, but were applied to the source code (language oriented metrics). The information these metrics need cannot be obtained until the algorithm and structure of the class are determined at the end of design the phase. However, the estimation of the fault-proneness in the early design phase helps in allocating effort for fixing the faults. In order to help assess quality of the software product early on during the development process, we have to give particular emphasis to the measurement of product design artifacts. Most of these studies have used static analysis of code and design documents to quantify the internal software properties. To deal with software failure we need to deal with dynamic behavior specification rather than static designs. Despite the rich body of research and practice in developing OO metrics, there has been less emphasis on dynamic metrics. Dynamic metrics are used to measure the dynamic behavior of a system based on the premise that active components are sources of failures [Lake, 1994]. The complex dynamic behavior of many real-time applications [Selic, 94] motivates a shift in interest from traditional static OO metrics to dynamics metrics.

### 3.1.3 Dynamic Metrics

Briand et al [Briand, 2000] indicated that measures of structural design properties, such as coupling or complexity, are widely considered to be indicators of external quality attributes, such as reliability. Dynamic metrics (complexity metrics and coupling metrics)

are used to measure the dynamic behavior of a system based on the premise that active components are sources of failures [Lake, 1994].

**Dynamic complexity metrics**

The study of software complexity metrics has long been practiced by scientists and engineers. Several studies found a correlation between the number of faults and the complexity of the system [Purao, 2003], [Elemam, 1999]. The OO metrics are either based on the static view of the design; i.e., its class diagrams, or the source code of the classes. The metrics are used to evaluate the complexity of the design structure and, hence, are static metrics. Most of these complexity definitions deal with the program at rest. However, the level of exposure of a module is a function of its execution environment. Hence, dynamic complexity [22] evolved as a measure of complexity of the subset of code that is actually executed. Dynamic complexity was discussed by Munson et al [Munson, 1996]. They emphasize that it is essential not only to consider complex modules but also how frequently they are executed. They define execution profiles for modules that reflect what percentage of time a module is executing, and thus derived functional complexity and operational complexity as dynamic complexity metrics. Toshihiro et al [Toshihiro, 1999] proposed a method to estimate the fault-proneness of the class in the early phase, using complexity metrics for object-oriented software. Ammar et al [Ammar, 1997] used Colored Petri Nets models to measure dynamic complexity of software systems using simulation reports. Poel et al [Poel, 2000] proposed a measures for a specific aspect of the dynamic behavior of objects i.e., life cycle complexity using the so-called distance-based approach. In chapter 4 we extend the dynamic complexity metrics to measure the quality of object-

oriented designs. Our approach is based on static analysis of UML statecharts that captures the behavior of the UML model.

**Dynamic Coupling Metrics**

Several studies have identified clear empirical relationships between class-level coupling and the fault-proneness of the classes. Measures of coupling have been shown to be reasonably accurate indicators of external quality attributes [Harrison, 2000]. In his research [Aaron, 1998] Aaron et al concluded that coupling metrics may be good predictors for run-time failures of a software product. Elemam et al [El-emam, 2001b] performed a validation study of object oriented design metrics. The objective of this validation was to determine which of these metrics was associated with fault-proneness. They conclude that many coupling measures are strongly associated with fault-proneness.

Coupling between components provides important information for identifying possible sources of exporting errors and identifying tightly coupled components. Moser et al [Moser,1997] introduced a model for measuring coupling and cohesion; this model is defined at the class level, only it does not reflect the dynamic behavior of the object. Li et al [Li, 1993] proposed the Message Passing Coupling (MPC) as a measure of coupling between classes which is defined as the number of "send" statements that are passed from one class to other classes. But counting the number of send statements does not reflect the actual number (frequency) of execution of that send statement. All these coupling measures are based on a static view of the software system. To be accurate and cost effective, there is a need to measure the dynamic interaction between software components early on in the software design phase.

The problem of measuring dynamic interaction between objects is addressed in [Poels, 2000]. The author proposed object-event association matrix to measure dynamic behavior of objects. Arisholm [Arisholm, 2002] proposed dynamic coupling measures, and described how dynamic coupling can be calculated by tracing the flow of messages between objects at run-time. It is impossible to be applied early on in the design phase; it is language and code based metrics. Lilli et al [Lilli, 2000] proposed a coupling metrics based on UML models. These metrics do not consider the dynamic behavior of the system. Aaron et al [Aaron, 1998] indicated that coupling metrics is a good predictor for run-time failure of the software product. Dynamic coupling could therefore be a good indicator for connector failure. Yacoub et al [Yacoub, 2000] defined dynamic coupling metrics based on UML, the execution of UML simulation model. It is tedious and time consuming to build a simulation model for the software system to measure coupling between components.

In the next chapter we explore the proposed architecture level dynamic coupling metrics as measures of coupling between components. These dynamic metrics are automatically extracted from UML visual static models.

### 3.2 Severity assessment review

Severity assessment is a procedure by which the severity of a software architectural element (component/connector) is estimated and ranked according to the consequences of failure. According to MIL_STD_1629A [Mil, 1986], severity considers the worst case consequence of a failure, determined by the degree of injury, property damage, system damage, and mission loss that could eventually occur.

To study the severity of software failure, we propose the Architecture Level Severity Analysis Methodology based on the UML model and hazard analysis techniques.

In this section, we review the hazard analysis techniques used with the proposed severity methodology, the suitability of UML as a foundation for severity assessment, and the related work on severity analysis.

### 3.2.1 Hazard analysis techniques

Hazard analysis techniques have been widely used in the development and deployment of safety critical systems that involve computer software [Francesmary 1997], [Heimdahl, 1996], [Leveson 1991], [Lutz 1993], [Ratan 1996]. Originally, a hazard analysis process is a systematic process for identifying, assessing and controlling hazards. The objectives of using any software hazard analysis technique are to identify and correct deficiencies and to provide information for the necessary safeguards. Software hazard analyses are used to change the software architecture or design, and to identify those portions of the software which require increased attention to quality. Software hazard analysis should be performed within the context of the overall system design, for those attributes of the system design that contribute to the system's ability to perform the assigned tasks.

Hazard analysis techniques such as Hazard and Operability Studies (HAZOP) [Heimdahl 1996], Failure Mode and Effects Analysis (FMEA) [NASA], Failure Modes Effects and Criticality Analysis (FMECA) [NASA, 1995 ], Fault Tree Analysis (FTA) [Allenby 2001], Functional Hazard Assessment (FHA) [Allenby, 2001], Event Tree Analysis (ETA) [NASA 1995], and Functional Failure Analysis (FFA) [Papadopoulos

1999] have demonstrated their value in a variety of contexts over the years, and they are still widely practiced by safety engineers.

HAZOP is a technique for identifying and analyzing the hazards and operation concerns of a system. In [Zhu, 2002] Hong adapted HAZOP and used it for quality modeling software process. McDermid and Pumfrey [McDermid, 1994] proposed a HAZOP analysis for identifying the hazardous failure modes of structural software model. In [Bishop, 2002], Bishop proposed a HAZOP based approach to support the justification of Commercial Off-The-Shelf software (COTS) used in a safety-related system. Hussey in [Hussey, 2000] was concerned with providing methods for analyzing safety-critical interactive systems to detect design defects that would reduce system safety. The approach presented is essentially a variant of the HAZOP. Bishop et al [Bishop 2002], described the Software Criticality Analysis approach that was developed using HAZOP. A HAZOP Study investigates the interactions between components and is carried out by a team. It is suitable for initial phases of the design.

The FMEA is an inductive qualitative method of analysis that enables us to identify elements having significant effects on system function in considered application. FMEA evaluates the ways component of the system can fail and the effects these failures can have. In a FMEA, each individual failure is considered as an independent occurrence with no relation to other failures in the system. In [Zhu, 2002] FMEA is used for the analysis of software and information systems to construct quality models of information systems. Cichocki et al [Cichocki, 2000] demonstrate how object oriented modeling, extended with formal notations, is used to model a problem related to computer based railway signaling in order to support FMEA. FMEA investigates the failures of the components themselves and

is often performed by an individual [Chudleigh, 1995]. FMEA identifies single failure modes that either directly result in or contribute significantly to system failure.

FTA is a top-down method used to identify failure causes [NASA, 1995]. FTA is primarily a means for analyzing causes of hazards, not identifying them. The process of analyzing causes is documented in one or more fault trees. It should be noted that a fault tree is not a model of the system or even a model of the ways in which the system could fail. It is rather a depiction of the logical interrelationships of the basic events that may lead to a particular undesired event. FTA is used to analysis a software hazard [Papadopoulos, 1999]. Papadopoulos [Papadopoulos, 2001] summarizes the FTA synthesis concept and discusses its application in the course of a continuous life-cycle safety assessment process. Traverson in [Traverson, 1998] proposed a formal hazard analysis technique using FTA to help designers understand the software requirements. They investigated how the results of one safety analysis technique, FTA, are interpreted as software safety requirements to be used in the program design process. Leveson et al [Leveson, 1999], proposed an approach for the incorporation of safety analysis methods in the software development process by using FTA for assessing safety properties of software. Hansen et al [Hansen, 1998] investigated, how the results of one fault trees are interpreted as software safety requirements to be used in the program design process. Liu et al [Liu, 1996] proposed a model-based approach for safety analysis using fault trees. Sere et al [Sere, 1997] used the results of FTA as a source of the formulation of requirements that the embedded software should meet. The approach proposed by Clarke and Clarke et al [Clarke, 1993] is based on representing the weakest precondition of a program as a fault tree. FHA approaches the analysis of the top-level design from the

functional viewpoint [ARP-4761 1996]. This technique aims to identify which functions of the system contribute to hazards, thus assigning them a criticality level. FHA was developed by the aerospace industry to bridge the gap between hardware and software hazard analysis, since functions are generally identified without specific implementations. It requires domain specific knowledge to produce meaningful results from Functional Hazard Analysis. The FHA method is used for identifying safety hazards at a functional level. It is a powerful method as it gives important input when structuring the requirements. In [Papadopoulos 1999] Papadopoulos developed the FFA method by extending FHA with guide words used in [McDermid, 2000] similar to the guide words used in HAZOP [McDermid, 1995].

The ETA [NASA, 1995] is a technique by which the system safety engineer can evaluate possible outcomes using a type of logic tree. It is an inductive logic method for identifying the various possible outcomes of a given initiating event. ETA is appropriate only after most of the design is complete. Thus, it has been used primarily to evaluate existing designs. In [Lindsay, 2000] Lindsay used ETA for hazard identification and analysis. A hazard analysis must be performed early in the software design, before any concept or design solutions exist, to avoid costly design iterations. The value of early hazard analysis has been thoroughly discussed by Leveson in [Leveson, 1995].

Automation has become increasingly more important [Gallow, 2002]. As systems become more complex, it is progressively less tractable to carry out hazard analyses manually. In computer HAZOP, the analysis remains manual activity in which analysts are called to identify and relate hazards by examining data flows in software architecture. As systems become more complex, manually performed hazard analyses become tedious,

error prone, and time-consuming [Lindsay, 2000]. A hazard analysis technique must be easy to automate or semi automate.

As the complexity of modern software systems increases, using one hazard analysis technique at different stages of the design lifecycle is becoming increasingly more problematic [Papadopoulos 1999], [Lindsay 2000]. These techniques assume different design representations which reflect different levels of abstraction in the system design. While, for example, FFA requires only abstract functional descriptions, HAZOP and FMEA require architectural designs of increasing detail and complexity. We must find a methodology to guarantee the consistency of the design as it evolves in the course of the lifecycle. All of this motivates the needs for a new hazard analysis technique.

We integrate more than one classical hazard analysis technique introducing a new technique that could be implemented early on in the design phase. This hazard analysis technique is used for the severity analysis and it could be automated because it is based on data that could be collected as well as analysis from UML diagrams [Hassan, 2003C].

### 3.2.2 UML and hazard analysis

In this section we discuss UML as a foundation for a new hazard analysis technique. To develop a hazard analysis technique, the relationship between the system and the environment should be defined sufficiently clearly so it could be possible to identify how system failures may cause harm (i.e., the system boundary is clear). The UML use case is used to capture requirements in early design phases. Thus, use cases are recommended as input to the FFA, to give a good picture of possible functional failures in a system [Johannessen, 2001]. In [Wedde, 1999] Wedde proposed the use of HAZOP UML based software system design.

For successful hazard analysis technique, the operation of the system has to be defined sufficiently clearly to indicate how normal or abnormal operation might cause system failure, and how the consequences of the system failure depend upon the operational state or mode. Because UML diagrams capture the dynamic behavior of the system it is a good candidate for a successful hazard analysis technique. Using UML [Atkinson 1998] as a foundation for hazard analysis will aid in understanding the system behavior before detailed design. By using UML models as a basis for the hazard analysis technique, system failures could be designed out of the system before it is developed and used.

The use of UML as a foundation of a new hazard analysis technique is motivated be the fact that UML models represent many level of abstraction of the system and capture its dynamic and static behavior.

The next section is the proposed severity analysis methodology which is based on UML artifacts. The methodology

It automates and integrates more than one hazard analysis technique in order to assess the severity of each architectural element (component/connector) of the software as well as the system level severity.

### 3.2.3. Severity analysis

In [Sherer, 1989] Sherer proposed a methodology to estimate the consequences of software failure caused by faults in different software modules. Sherer used one hazard analysis technique FTA and software operational profile to estimate the cost of failure for every software module. This is a complex process to be automated and it was applied in the later phase of software development life cycle. Yacoub et al [Yacoub, 2002] used

FMEA to assess the severity of software (components/connectors) failure. Using one hazard analysis technique for severity analysis usually fails to offer a coherent and complete picture of the ways in which low-level component failures contribute to hazardous malfunctions of the system. Yacoub and Sherer used one hazard analysis technique. Hazard analysis techniques assume different design representations which reflect different levels of abstraction in the system design. While, for example, FFA requires only abstract functional descriptions, HAZOP and FMEA require architectural designs of increasing detail and complexity. It is not necessarily enough to use only one analysis technique [Allenby, 2001]. Often a combination of more than one technique, based on UML, should be used in order to gain a more complete understanding of the system [Hassan, 2003b].

In chapter 6 we propose a methodology for severity analysis based on UML diagrams. The proposed methodology is based on a number of classical hazard techniques such as FFA, FMEA and FTA. It automates and integrates these techniques in order to assess the severity of each architectural element (component/connector) of the software.

### 3.3 Risk Assessment

A wide range of traditional fault-proneness prediction models has been proposed for assessing the fault proneness of a software system. These models use complexity and size metrics to predict software components fault proneness. These models are quantitative models that can be used to predict which software components are high risk keeping in mind that the definition of a high risk component varies depending on the context. For example, a high risk component may be one that contains any faults found during testing

[Briand, 1993a], [Lanubile, 1997], [Harrison, 1988], or one that contains any faults found during operation [Khoshgoftaar, 1999], or one that is costly to correct after an error has been found [Almeida, 1998], [Basili, 1997], [Briand, 1993b]. Some models make binary predictions as to whether a software component is faulty or not-faulty [El-Emam, 1999], [El-Emam, 2001a], [Khoshgoftaar, 1999], [Lanubile, 1997]. These models are also used for ranking software components according to risk proneness.

Silke et al [Silke, 1999] proposed a model to estimate the probability of failure of a software system consisting of components. Briand et al [Briand, 1998] used multivariate logistic regression to build a prediction model for the fault-proneness of classes. Ping et al [Ping, 2000] proposed a statistical technique of mixture model analysis as a tool for early prediction of fault-prone software modules. Lanubile et al [Lanubile, 1995] proposed a model for identifying high/low-risk of software components based on software complexity metrics. Maurizio et al [Maurizio, 1997] proposed a model to predict defect-prone software modules in a software system based on its complexity. Hochman et al [Hochman, 1996] applied the genetic algorithm to developing optimal or near optimal back propagation neural networks for fault-prone/not-fault-prone classification of software modules. Toshihiro et al [Toshihiro, 1999] proposed a method to estimate the fault-proneness of the class in the early phase, using several complexity metrics for object-oriented software. Wong et al [Wong, 1998] proposed hybrid metrics to identify fault-prone software modules. Benlarbi [Benlarbi, 1999] proposed a model that serves as an early predictors of classes that contain faults.

But all of these traditional models do not consider the fact that the consequences of various software failures caused by faults will be very different. Science software failures

have different consequences; any measure of software fault proneness must include the measurement of the consequence of failure (severity).

Based on the definition of heuristic risk factor [Ammar, 1997] as a measure of risk, Yacoub et al [Yacoub, 2002] developed an architecture level risk assessment methodology of software system taking into consideration the severity of software failure. Yacoub used a set of dynamic metrics extracted from a simulation model of the UML artifacts and used the FMEA for severity assessment. But risk assessment based on UML simulation model is time consuming and tedious work. Using FMEA only for severity analysis is not enough as we mentioned earlier in "Hazard Analysis Review".

To help in developing risk based test plans, we propose a risk assessment methodology for the software architecture. This methodology is based on measurements that could be collected and analyzed early on in the software life cycle. It considers the severity of software failure as well as the probability of that failure. This methodology is performed integrating more than one hazard analysis technique based on dynamic specification of static UML model. To automate the proposed methodology we have proposed a tool support for the automation of this methodology.

### 3.4 Summary

This literature review indicates the absence of research for early software product risk assessment. Most of the work does not differentiate between risk assessment and fault proneness. It also indicates the lake of concrete methodologies for risk assessment early on the design phases. This literature review indicates that there is a need for clear methods for software severity analysis. It motivates us also to explore the dynamic metrics based on the UML artifacts.

# *Chapter 4*

## Dynamic Specifications Metrics using UML Static Models

### 4.1 Introduction

The fact that the usage of metrics in the analysis and design of object oriented OO software can help designers make better decisions is gaining relevance in the software measurement arena [Fenton, 2000]. Moreover, the necessity of having early indicators of external quality attributes, such as maintainability, based on early metrics is growing.

These metrics are based either on the static view of the design; i.e., its class diagrams, [Marcela, 2003] or the source code [Chidamber, 1994] of the classes. These static metrics deal with the class code or the structure of the class. These types of static metrics do not consider the class behavior or its execution environment. However, the level of exposure of a software component is a function of its execution environment. Hence, dynamic metrics [Khoshgoftaar, 1993] evolved as a measure of complexity of the subset of code that is actually executed. Dynamic metrics was discussed by Munson et al [Munson, 1996] for reliability assessment purposes. The authors emphasized that it is essential to consider complex modules and its frequency of execution, in addition to its complexity. Ammar et al [Ammar, 1997] extended dynamic metrics definitions to incorporate concurrency complexity. They further used Colored Petri Nets models to measure dynamic complexity of software systems using simulation reports. To measure the dynamic behavior of an object in OO system Poel et al [Poel, 1999] define a so-called distance-based approach, which describes and measures the dynamic behavior of objects. Yacoub et al [Yacoub, 1999] proposed a dynamic metrics (dynamic complexity and

dynamic coupling) based on the execution of UML simulation model of the software system. Using simulation model of the system to measure dynamic metrics is a more tedious and time consuming process than using UML models at system design phase.

In this chapter, we build up on metrics proposed in [Yacoub, 1999] to estimate the dynamic metrics (complexity and coupling) of a software architectural element (component/connector) early on the software design phase. This proposed approach [Hassan, 2001] is based on static analysis of UML diagrams of the software system not based on execution of the simulation models as proposed in [Yacoub, 1999]. The proposed dynamic complexity metrics could be used for the estimation of probability of software component/connector failure early on the design phase of the software life cycle [Katerina, 2003].

The software system design phase serves as the foundation for all software engineering steps that follow. Design phase is important in this respect, since many of the critical decisions stem from the product design. In the design phase, it's still cost-effective to modify the software design, and the software can also be described accurately enough to be measurable [Roger, 1997]. Our proposed dynamic metrics is based on UML diagrams developed at the design phase.

We proceed in this chapter as follows: Section 4.2 describes the dynamic complexity metrics, Section 4.3 describes the dynamic coupling metrics, Case study is presented in Section 4.4 and Section 4.5 is the conclusion.

**4.2 Dynamic Complexity Metrics**

Dynamic complexity metrics guide the process of identifying complex components. As a result, components could be ranked based its complexity. Components

in a system have a life cycle; they enter into an initial state after creation, when they are triggered (e.g. by messages, events, etc), and may change state. At some moment in time, they enter an ending state. The life of component in a system is a dynamic property. We need some kind of state transition model to measure component complexity during its life cycle [Poels, 1999]. The state machine view describes the dynamic behavior of components over the time by modeling the lifecycle of components. Each component is treated as an isolated entity that communicates with the rest of the world by detecting events and responding to them. A transition defines the response of a component in the state to the occurrence of an event. A statechart store information about component life cycle and component behavior [Booch, 1999]. Both the number of states in the state chart as well as the number of transitions can be measures for a component's dynamic complexity [Khoshgoftaar, 1999].

Based on state diagrams of the UML model, we could estimate dynamic complexity metrics. This metrics could be estimated during a specific scenario of interaction (sequence diagram). Based on a state chart of components interacted during sequence diagram, we could compute dynamic metrics for each component. Measurements could be extended for all scenarios under specific use case. Also use case and system level complexity could be estimated by providing the profile of operation of all scenarios and use cases.

### 4.2.1 Dynamic complexity of a component ($dco_i^x$)

In 1976 McCabe introduced cyclomatic complexity as a measure of program complexity [McCabe, 1976]. It is obtained from the control flow graph and defined as $CC = e - n + 2$, where $e$ is number of edges and $n$ is number of nodes in the control

flow graph. We propose a measure of component complexity similar to McCabe's cyclomatic complexity. However, in contrast to McCabe's cyclometric complexity, which is based on the control flow graph of the source code, the proposed metric for component's dynamic complexity is based on the UML state charts that are available during early stages of the software life cycle. The state chart of a component $i$ has a number of states and transition between these states that describe the dynamic behavior of the component. For each scenario $S_x$ a subset of all states of component $i$ are visited and a subset of all transitions is traversed. Let $C_i^x$ denote the subset of states for a component $i$ visited in the scenario $S_x$ and with $T_i^x$ the subset of transitions traversed in the state chart of component $i$ in the scenario $S_x$. The subset of states $C_i^x$ and the corresponding transitions $T_i^x$ are mapped into a control flow graph. The number of nodes in this graph is $c_i^x = \left| C_i^x \right|$; this number is the cardinality of $C_i^x$. Similarly, the number of edges in this graph is $t_i^x = \left| T_i^x \right|$; this number is the cardinality of $T_i^x$. It follows that the dynamic complexity $dco_i^x$ of component $i$ in scenario $S_x$ is defined as

$$dco_i^x = t_i^x - c_i^x + 2. \qquad 4\text{-}1$$

### 4.2.2 Normalized dynamic complexity of a component $(DOC_i^x)$

The normalized dynamic complexity $DOC_i^x$ of a component $i$ in scenario $S_x$ is obtained by normalizing the dynamic complexity $dco_i^x$ with respect to the sum of complexities for all active components in scenario $S_x$

$$DOC_i^x = \frac{dco_i^x}{\sum_{k \in S_x} dco_k^x} \qquad 4\text{-}2$$

where $DOC_i^x$ $(0 \leq DOC_i^x \leq 1)$ is the normalized complexity of the $i^{th}$ component in the scenario $S_x$. This normalized complexity is used as the probability of failure of a component [Hassan, 2001].

### 4.3 Dynamic Coupling Metrics

Coupling between components provides important information for identifying possible sources of exporting errors, identifying tightly coupled components, and testing interactions between components. The proposed dynamic coupling metrics extend the previous work in [Yacoub, 1999]; measurements are calculated for the design model during a specific scenario. Furthermore, they can be extended for all scenarios in all use cases of the system. The dynamic coupling of a connector is based on the number of messages that are carried by the connector.

Our approach in this part is to calculate the measurements directly from the UML scenario diagrams, not from execution of the simulation model [Yacoub, 1999]. Dynamic coupling metrics calculated from simulation models [Yacoub, 1999] are defined as export dynamic coupling and import dynamic coupling. The difference between export and import coupling is in the direction of data and control flow from/to a component in the architecture. Export coupling dealings with coupling as a component sends messages or data to other components. Import coupling dealings with coupling as a component receives messages or data from other components in the architecture. We use export dynamic coupling for this proposed dynamic coupling. Export coupling accounts for the

fact that an error in a currently executing component could be exported to the called component.

### 4.3.1 Normalized dynamic coupling of a connector ($EOC_{ij}^{x}$)

Define $EOC_{ij}^{x}$ as a measure of mutual coupling between two specific components ($i$ and $j$ ). Let denote with $MT_{ij}^{x}$ the set of messages sent from component $i$ to component $j$ during the execution of scenario $S_x$ and with $MT^{x}$ the set of all messages exchanged between all components active during the execution of scenario $S_x$. Then, we define the export coupling $EOC_{ij}^{x}$ from component $i$ to component $j$ in scenario $S_x$ as a ratio of the number of messages sent from $i$ to $j$ and the total number of messages exchanged in the scenario $S_x$.

$$EOC_{ij}^{x} = \frac{\left|MT_{ij}^{x}\right| \quad i, j \in S_x \ , i \neq j}{\left|MT^{x}\right|} \qquad 4\text{-}3$$

Where $EOC_{ij}^{x}$ ($0 \leq EOC_{ij}^{x} \leq 1$) is the normalized coupling for the connector between $i^{th}$ and $j^{th}$ components in the scenario $S_x$. Normalized dynamic coupling of a connector is used as the probability of failure of connector between components [Katerina, 2003].

## 4.4 Case study

In this section, we explain the specifications of Cardiac Pacemaker system. Also we show how the proposed methodology applied using this case study as illustrative example. Cardiac pacemaker is an implanted device that assists cardiac functions when the underlying pathologies lower intrinsic heartbeats. An error in the software operation

of the device can cause loss of a patient's life. This is an example of a critical real-time application. We use the UML Real-Time notion [Rational, 2001] to model the pacemaker. Figure 4.1 shows the components and connectors of the pacemaker in the capsule diagram [Yacoub, 1999]. The Figure also shows the input/output port to the Heart as an external component as well as the two input ports to the *ReedSwitch* and the *CoilDriver* components. A pacemaker can be programmed to operate in one of the five operational modes depending on which part of the heart is to be sensed and which part is to be paced.

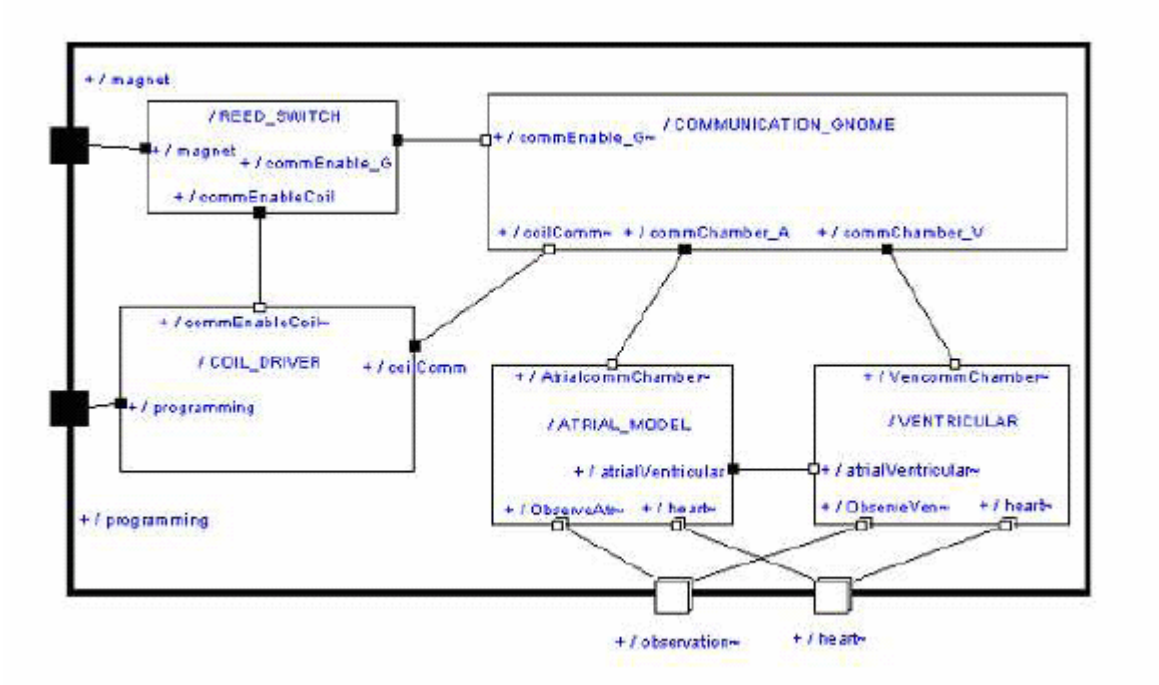



Figure 4.1 the components and connectors of the pacemaker in the capsule diagram

The main components of the cardiac pacemaker system are described as follows:

*ReedSwitch*(*RS*): A magnetically activated switch that must be closed before programming the device. The switch is used to avoid accidental programming by electric noise.

*CoilDriver*(*CD*): Receives/sends pulses from/to the device programmer. These pulses are counted and then interpreted as bit values of zero or one. These bits are then grouped into bytes and sent to the communication gnome. Positive and negative acknowledgments, as well as programming bits, are sent back to the programmer to confirm whether the device has been correctly programmed and the commands validated.

*CommunicationGnome*(*CG*): Receives bytes from the Coil Driver, verifies these bytes as commands, and sends the commands to the Ventricular and Atrial models. It sends the positive and negative acknowledgments to the Coil Driver to verify command processing.

*AtrialModel*(*AR*) and *VentricularModel*(*VT*): These two components are similar in operation. They both could pace the heart and/or sense the heartbeats. Once the pacemaker is programmed, the magnet is removed from the *RS*. The *AR* and *VT* communicate together without further intervention. Only battery decay or some medical maintenance reasons may force reprogramming.

### 4.4.1 The Use case model

The pacemaker runs in either a programming mode or in one of five operational modes. During programming, the programmer specifies the operation mode in which the device will work. The operation mode depends on whether the *A*trial (*A*), Ventricular (*V*) or both are being monitored or paced. The programmer also specifies whether the pacing is inhibited (*I*), triggered (*T*) or dual (*D*). For example in the *AVI* operation mode, the ***A***trial portion (*A*) of the heart is paced (shocked), the ***V***entricular portion (*V*) of the heart

is sensed (monitored) and the ***A***trial is only paced when a ***V***entricular sense does not occur (inhibited mode). The use case diagram of the pacemaker application is given in Figure 4.2. It presents the six use cases the two actors namely *doctor's programmer* and *patient's heart*. Each use case in Figure 4.2 is realized by at least one sequence diagram (i.e., scenario). Domain experts determine probabilities of occurrence of use cases and the scenarios within each use case. This can be done in a similar fashion as the estimation of the operational profile in the field of software reliability [Musa, 1996].

According to [Douglass, 2000], inhibit modes are more frequently used than the triggered mode [Douglass, 2000]. Also, the programming mode is executed significantly less frequently than the regular usage of the pacemaker in any of its operational modes. Hence, we assume the probabilities for programming use case of five operational use cases (*AVI, AAI, AAT, VVI* and *VVT*) as given in table 4.1.

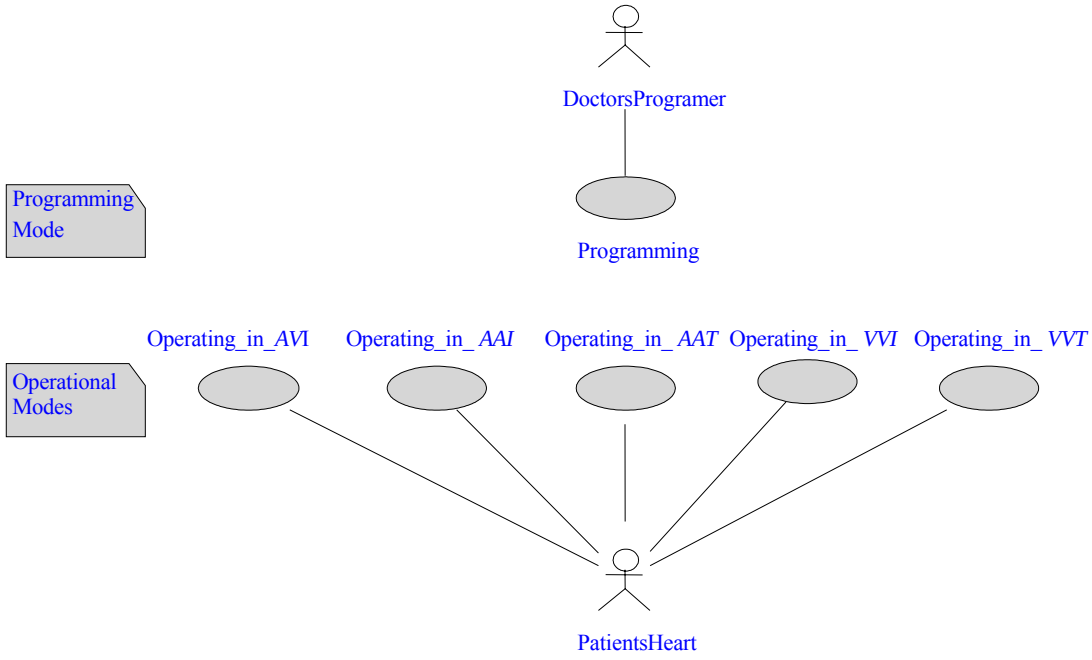



Figure 4.2 Pace maker use case diagram

| Use case | *Programming* | *AVI* | *AAI* | *VVI* | *AAT* | *VVT* |
|---|---|---|---|---|---|---|
| Probability | 0.01 | 0.29 | 0.20 | 0.20 | 0.15 | 0.15 |

Table 4.1 Probabilities of the use cases executions

In the programming use case, the programmer interacts with the *RS* and *CD* components to input a set of 8 bits specifying an operation mode for the pacemaker. This byte is received by the *CG* component, which, in turn, sets the operation mode of the AR and *VT* components to one of five modes (or use cases): *AVI, AAI, AAT, VVI,* and *VVT*. Appendix A shows a scenario from the *AVI* use case in which the *VT* keeps sensing the heart and the *AR* paces the heart whenever a heart beat is not sensed. As in all scenarios, a refractory period is then in effect after every pace. Every use case is mapped to one scenario. The first is *Programming* scenario in which the programmer sets the operation mode of the device. The programmer applies a magnet to enable communication with the device, and then sends pulses to the device which in turn interpret these pulses into programming bits. The device then sends back the data to acknowledge valid/invalid program. The second scenario is the *AVI* scenario. In this scenario, the *VT* component monitors the heart. When a heart beat is not sensed, the *AR* component paces the heart and a refractory period is then in effect. The third (fourth) scenario is the *VVI* (*AAI*) scenario in which the *VT* component (*AR* component) paces the heart when it does not sense any heart pulse. The fifth (sixth) scenario is the *VVT* (*AAT*) in which the *VT* component (*AR* component) continuously paces the heart.

The detailed scenario diagrams and the rest of the results of the pacemaker are shown in appendix A.

Figure 4.3 shows the sequence diagram of the *programming* scenario of the pacemaker system, which we use to illustrate dynamic complexity of components interact during this scenarios and dynamic coupling of connectors between components in this scenarios.

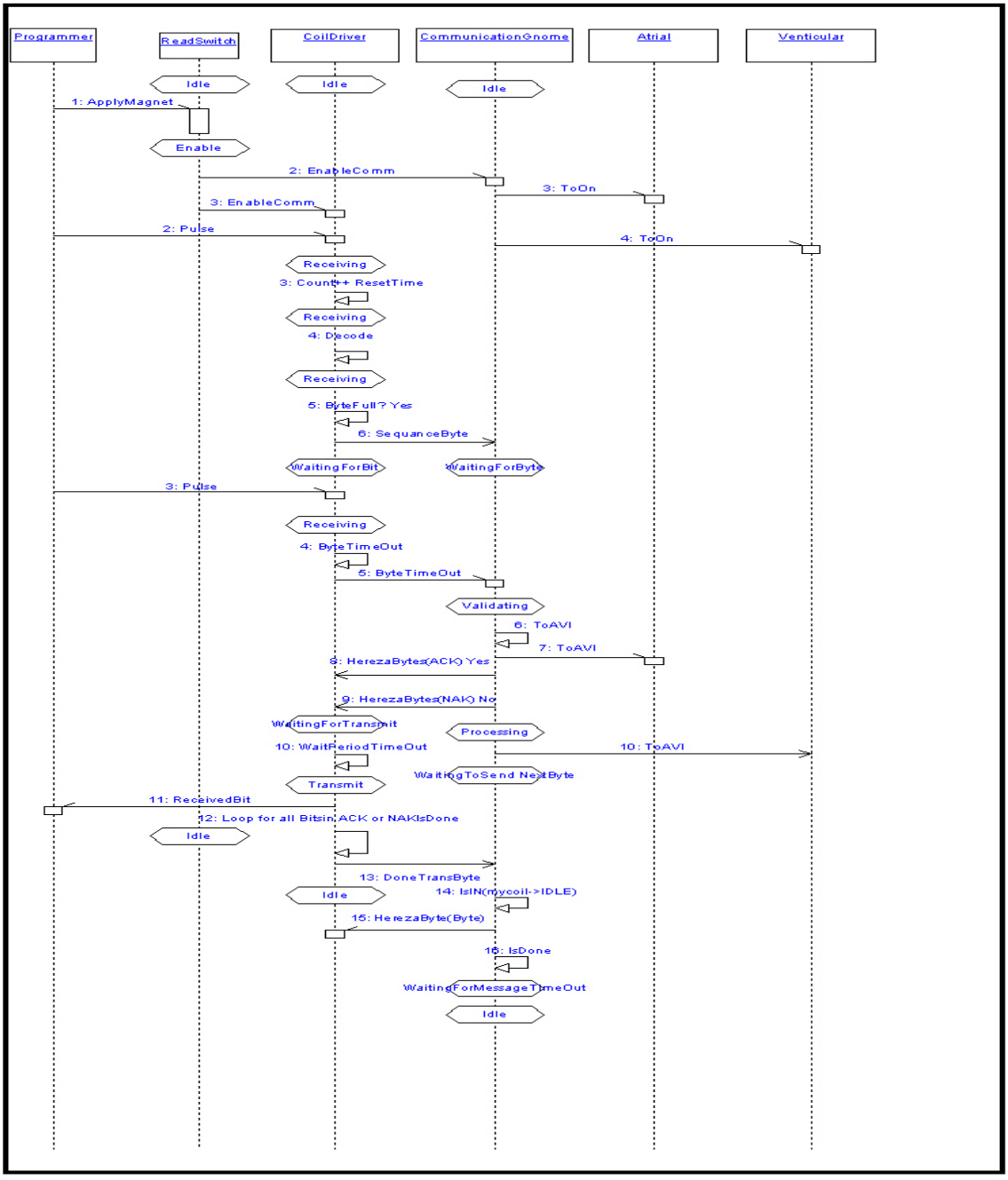



Figure 4.3 Sequence diagram of the *programming* scenario

### 4.4.2 Dynamic complexity of components

As described in section 4.2 we use a measure of component complexity similar to McCabe's cyclomatic complexity. However, in contrast to McCabe's cyclometric

complexity, which is based on the control flow graph of the source code, the state chart of each component $i$ has a number of states and transition between these states that describe the dynamic behavior of the component. For each scenario $S_x$ a subset of all states of component $i$ are visited and a subset of all transitions is traversed. Figure 4.4 shows a subset of states of *CD* component in the *programming* scenario. The control flow graph of the *CD* component in the *programming* scenario is presented in Figure 4.4. The dynamic complexity of this graph is evaluated using equation 4.1 and normalized with respect to the sum of complexities of all active components in this scenario (*RS*, *CD*, and *CG*) using equation 4.2.

Table 4.2 shows the normalized dynamic complexity for all components active in the *programming* scenario.

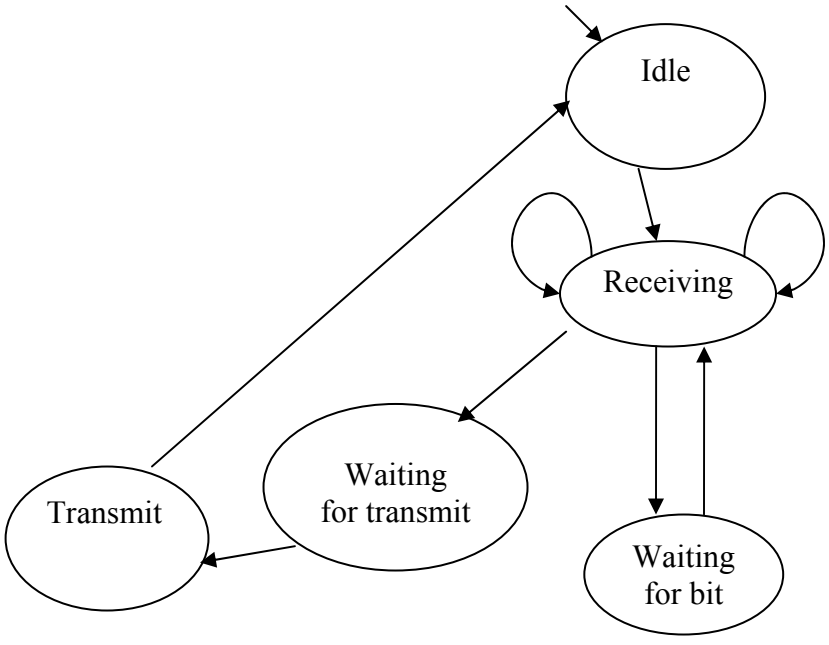



Figure 4.4 control flow graph for states and transitions of the *CD* component in the *programming* scenario

| **Component** | $DOC_i^x$ (Dynamic Complexity) |
|---|---|
| *CD* | 0.5 |
| *RS* | 0.2 |
| *CG* | 0.3 |

Table 4.2 Normalized dynamic complexity of components in the *programming* scenario

Dynamic complexity of all components interacts within the *pacemaker* scenarios are given in Table 4-3.

| Scenario Name | Component Dynamic Complexity | | | | |
|---|---|---|---|---|---|
| | RS | CD | CG | AR | VT |
| *Programming* | 0.2 | 0.5 | 0.3 | 0.0 | 0.0 |
| *AVI* | 0.0 | 0.0 | 0.00017 | 0.60135 | 0.34837 |
| *AAI* | 0.0 | 0.0 | 0.0009 | 0.999 | 0.0 |
| *VVI* | 0.0 | 0.0 | 0.0009 | 0.0 | 0.999 |
| *AAT* | 0.0 | 0.0 | 0.0005 | 0.9995 | 0.0 |
| *VVT* | 0.0 | 0.0 | 0.0005 | 0.0 | 0.9995 |

Table 4.3 component dynamic complexity for all scenarios in *pacemaker*.

Figure 4.5 shows the 3D pars for components dynamic complexity Vs scenarios in the pacemaker system.

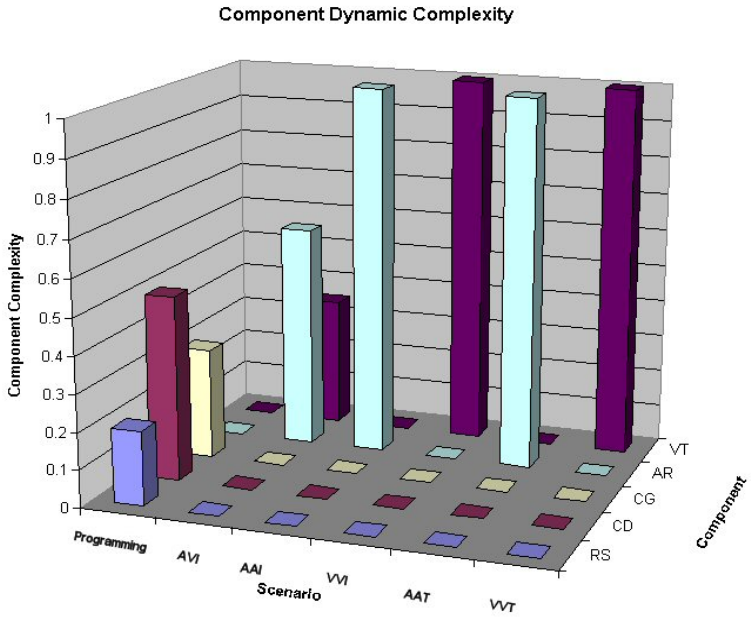



Figure 4.5 3D pars for components dynamic complexity Vs scenarios

of the *pacemaker*.

We conclude from table 4.3 and Figure 4.5 that the dynamic complexity of the component *AR* and *VT* are significantly higher than those for other components in more than one scenario. This is due to the fact that those two components are active and executing most of the time.

### 4.4.3 Dynamic coupling of connectors

Dynamic coupling metrics are calculated for active connectors during execution of a specific scenario. We compute these metrics directly from the UML sequence diagrams by applying the set of formulas given in section 4.3. Based on equation 4.3, we calculate dynamic coupling of connectors in the *programming* scenario shown in Table 4.4 shows the dynamic coupling matrix for the connectors in this scenario.

We use the matrix representation for coupling where rows and columns are indexed by components and the off-diagonal matrix cells represent coupling between the two components of the corresponding row and column [Hassan, 2001]. The row index indicates the sending component, while the column index indicates the receiving component. For example, the cell with row=*RS* and column=*CD* is the export coupling value from *RS* to *CD*. On the other side, the cell with row=*CD* and column=*RS* is the export coupling value from *CD* to *RS*. For example, the value along the row *RS* and the column *CD* in table 4.4 is 0.125. This is read as dynamic coupling of the connector from *RS* to *CD*.

| | *RS* | *CD* | *CG* |
|---|---|---|---|
| *RS* | 0 | 0.125 | 0.125 |
| *CD* | 0 | 0 | 0.375 |
| *CG* | 0 | 0.375 | 0 |

Table 4.4 Dynamic coupling of connectors in the *programming* scenario

Table 4-5 and Figure 4.6 show the results from applying the proposed dynamic coupling metrics for all connectors for all scenarios of the *pacemaker* system.

| Scenario Name | Connector Dynamic Coupling | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | RS-CD | RS-CG | CD-CG | CG-CD | CG-AR | CG-VT | AR-VT | VT-AR |
| *Programming* | 0.125 | 0.125 | 0.375 | 0.375 | 0.0 | 0.0 | 0.0 | 0.0 |
| *AVI* | 0.0 | 0.0 | 0.0 | 0.0 | 3.90E-04 | 3.90E-04 | 0.097 | 0.9 |
| *AAI* | 0.0 | 0.0 | 0.0 | 0.0 | 1 | 0.0 | 0.0 | 0.0 |
| *VVI* | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 | 0.0 | 0.0 |
| *AAT* | 0.0 | 0.0 | 0.0 | 0.0 | 1 | 0.0 | 0.0 | 0.0 |
| *VVT* | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1 | 0.0 | 0.0 |

Table 4.5 dynamic coupling for every connector in each scenario for pacemaker system

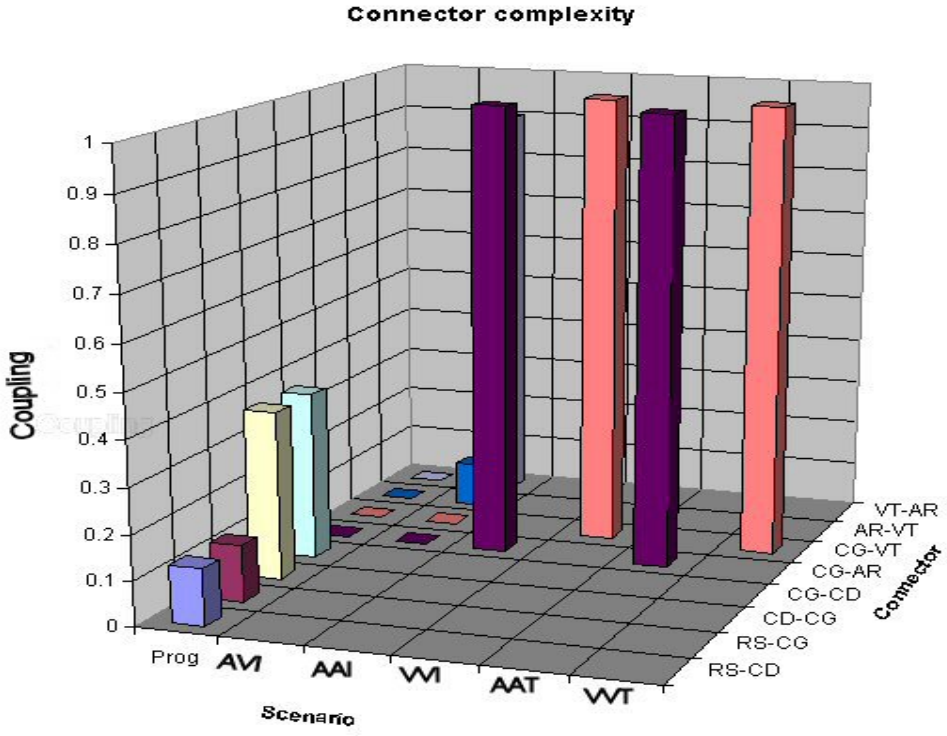



Figure 4.6 3D par for connectors dynamic coupling Vs scenarios

These results have identified that components *CG, AR, VT* are the most dynamic coupled components; this is due to the fact that these components are the most active components in more than one scenario and these components send messages to each other most of the time. These components usually communicate with each other and with

the heart for sensing and monitoring. These components control the operational processing of the system.

### 4.4.4 Validating the Dynamic metrics

In order to validate the proposed dynamic metrics we compared the predicted results derived from the proposed dynamic metrics with those derived from the simulation model built in [Yacoub, 1999] for the same case study.

The idea of using simulation models to evaluate predictive techniques is explored in [Shepperd, 2001]; Shepperd proposed the using a simulation model results for evaluating four predictive techniques.

To compare the results obtained by the proposed metrics with the results obtained from the simulation model [Yacoub, 1999]; we average the components/connectors dynamic metrics for all scenarios in the pacemaker system based on the operational profile of the pacemaker system. Table 4.6 and Figure 4.7 show the system level component dynamic complexity and the simulation results obtained in [Yacoub, 1999]. The correlation of the result obtained by the two methods is shown.

| | System level component dynamic complexity | | | | |
|---|---|---|---|---|---|
| | RS | CD | CG | AR | VT |
| Dynamic complexity normalized to the maximum one | 0.0038 | 0.0095 | 0.0068 | 1 | 0.86 |
| Simulation Model results Normalized to the maximum one | 0.002 | 0.013 | 0.005 | 1 | 0.963 |
| Correlation = 0.99732405 | | | | | |

Table 4.6 system level component dynamic complexity vs. simulation model results

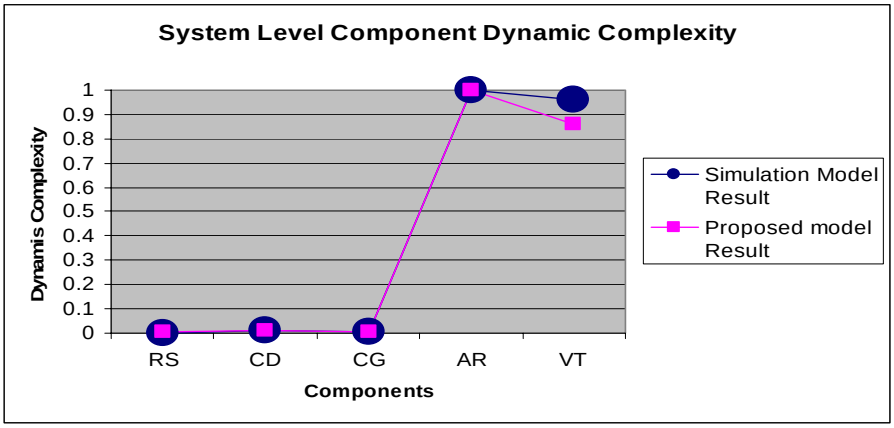



Figure 4.7 System level component dynamic complexities vs. simulation model result

Table 4-7 and Figure 4.8 show the system level results derived from the proposed dynamic coupling and the results derived from the simulation.

| | System level dynamic coupling | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | RS-CD | RS-CG | CD-CG | CG-CD | CG-AR | CG-VT | AR-VT | VT-AR |
| Dynamic coupling normalized to the maximum one | 0.00357 | 0.00357 | 0.0107 | 0.0107 | 1 | 1 | 0.0803 | 0.7455 |
| Simulation Model result Normalized to the maximum one | 0.0014 | 0.0014 | 0.003 | 0.002 | 0.0014 | 0.0014 | 0.25 | 0.27 |
| Correlation = 0.088739 | | | | | | | | |

Table 4.7 System level connector dynamic coupling Vs. simulation model result

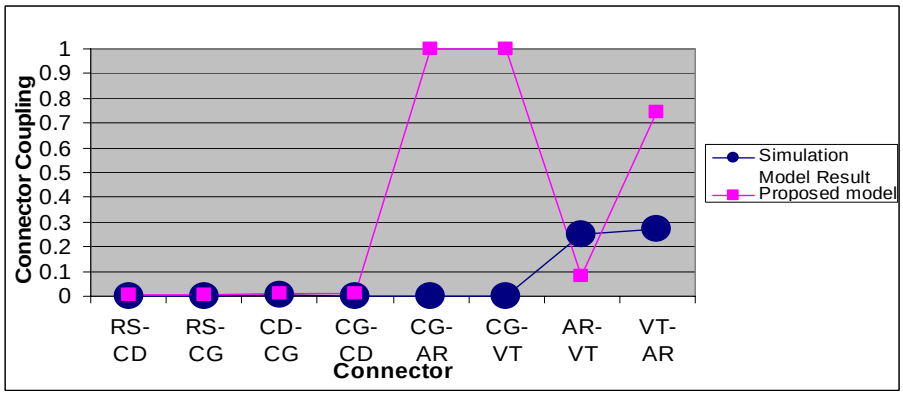



Figure 4.8 System level connector dynamic coupling Vs. simulation model result

From Figure 4.7 and Figure 4.8 the comparison shows a high correlation for dynamic complexity but poor correlation for dynamic coupling. This may be due to the fact that Yacoub [Yacoub, 1999] counted the message exchange between heart and programmer in his calculation but this is not the case in our calculation. We do not consider the *heart* and *programmer* as a part of our software system; we consider *heart* and *programmer* as external actors. They are not part of the pacemaker software system.

## 4.5 Conclusion

In this chapter we define two dynamic metrics (dynamic complexity/dynamic coupling) for software architectural element (component/connector). Measurements using these proposed metrics can be obtained at early development phases from UML models. The metrics are applied to a pacemaker case study and measurements are compared to results from execution models [Yacoub, 1999] to show the validity of the metrics. The comparison shows a high correlation of the proposed method with the simulation results obtained in [Yacoub, 1999] for dynamic complexity. The results of dynamic coupling show low correlation with [Yacoub, 1999]. This low correlation could be because Yacoub [Yacoub, 1999] considers actors at every scenario as a part of the software system. He considers the messages exchanged between the actors and the system as a part of all messages exchanged in each scenario. We consider the actors at every scenario as external entity and actors are not part of our software system. We do not consider the messages exchanged between the actors and the system as part of our calculations. In future work, we could validate coupling metrics using another case study. We could also explore the dependency between static and dynamic metrics, and empirically validate the proposed metrics and their correlation with design quality attributes.

# *Chapter 5*

## UML Based Severity Analysis Methodology

### 5.1 Introduction

Severity assessment is a procedure by which the severity of failures of software architectural element (component/connector) is estimated and ranked accordingly to the consequences of these failures. In the MIL_STD_1629A standard [MIL_STD, 1986], severity considers the worst case consequence of a failure, determined by the degree of injury, property damage, system damage, and mission loss that could eventually occur.

Considering the severity of software failures will help in allocating development and testing resources [Rosenberg, 1999]. Some software modules may be tested more intensively than others based on the severity of failure weighted by the probability of failure. In [Katerina, 2003], we proposed an architectural level software risk assessment methodology. The proposed risk methodology combined the probability of software failure with the severity of this failure to estimate the risk factor of software architectural element early on the software design phase. The probability of failure is estimated based on software dynamic metrics [Hassan, 2001]. In this report we propose a methodology for estimating severity of failures of software components/connectors as well as severity of failures of system execution scenarios early on during software development.

Sherer in [Sherer, 1988] proposed a methodology to estimate the consequences of software failure caused by faults in different software modules. Sherer used FTA and software operational profile to estimate the cost of failure for every software module. This is a complex process to be automated and it was applied at the code level in the later

phases of software development. Yacoub et al [Yacoub, 2002] used FMEA to assess the severity of software (components/connectors) failure. Using one hazard analysis technique for severity analysis usually fails to offer a coherent and complete picture of the ways in which low-level component failures contribute to hazardous malfunctions of the system. Hazard analysis techniques assume different design representations which reflect different levels of abstraction in the system design. While, for example, FFA requires only abstract functional descriptions, HAZOP [McDermid, 1995] and FMEA require architectural designs of increasing detail and complexity. As shown by Allenby and Kelly in [Allenby, 2001] it is not enough to only use one severity analysis technique in complex systems. Often a combination of more than one technique should be used to get a more complete understanding of the system. The suitability of UML [Boosh, 1999] to be the specification language for severity analysis using more than one classical hazard analysis method was briefly discussed in [Hassan, 2003b], [Guiochet, 2003]. We propose a methodology for severity analysis of software systems at the early phases of development based on UML. This chapter is organized as follows, Section 5.2 presents the proposed methodology; Section 5.3 presents the illustrative case study; and Section 5.4 provides the conclusions and future work.

## 5.2 The proposed severity analysis methodology

The proposed severity analysis methodology starts early in the development phase with FFA which uses system level scenario diagrams as an input to identify all system level hazards [Johannessen, 2001]. This high level FFA analysis gives us a comprehensive view of the ways in which the system could fail. System level failures arise as a result of failures or malfunctions of lower level components/connectors.

Therefore we apply FMEA as the second step of the process, at the level of components and connectors using UML sequence diagrams to determine their failure modes and cost of failure for each failure mode. We use FTA as a third step to define a relationship between failures of individual architecture elements (component/connector) and a failure of the system. The system hazards identified in step 1 will be used as top events in the fault tree and the basic events are the failure modes identified in step 2. The fourth step of this process is to develop the cost of failure graph [Sherer, 1988] to estimate cost of failure for each execution scenario and every component/connector in the scenario. The final step is to map estimated cost of failure of scenario and each component/connector to a severity rank using the cost severity graph introduced in [Kmenta, 2000].

Figure 5.1 shows the schematic diagram of the proposed severity analysis methodology, and the steps of the methodology for a given scenario are summarized as follows:

1. Identify system hazards (states of the system that can contribute to accidents and mishaps) by performing FFA [Carpenter, 1999].
2. Identify components/connectors failure modes by performing FMEA [MIL_STD, 1986].
3. Construct a detailed cause and effect model that records how failures propagate from components/connectors level through the system level by using FTA. This step combines the outputs from step 1 and step 2.
4. Develop the Cost of Failure Graph to estimate cost of failure of a given execution scenario and each component/connector in this scenario [Sherer, 1988].

5. Estimate the severity of each component/connector and system scenario using cost of failure graph [Sherer, 1988] and cost severity graph [Kmenta, 2000].

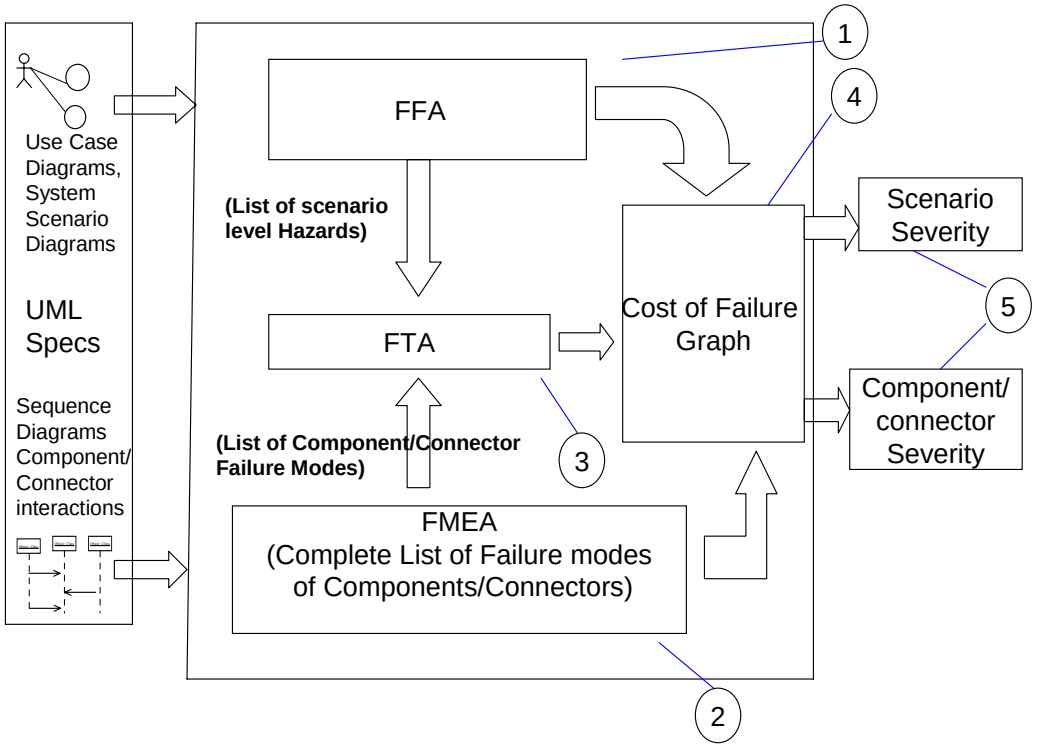



Figure 5.1 severity analysis methodology schematic diagram

### 5.2.1 Functional Failure Analysis (FFA)

Figure 5.2 shows a UML use case diagram for a system ***S***, where actor $Act_1$ performs the two Use Cases $Uc_1$ and $Uc_2$ through association $Ass_1$ and $Ass_2$, and actor $Act_2$ performs the Use Case $Uc_1$ through $Ass_3$. Figure 5.3 shows a high level system sequence diagram [Carpenter, 1999] which describes one scenario of the Use Case $Uc_1$ showing the interactions between actors $Act_1$, $Act_2$ and the system through input and output events. Events like $E_{11s}$, $E_{21s}$, $E_{3s1}$, and $E_{4s1}$ are the events between the system ***S*** and $Act_1$. Events $E_{1s2}$, $E_{22s}$, and $E_{3s2}$ are events between the System ***S*** and the $Act_2$. The system states are $S_1$, $S_2$, $S_3$, and $S_4$ ,which are the states of the system after receiving or

sending event to the external actors (***Act*$_1$**, ***Act*$_2$** ) The input events (***E*$_{11s}$**, ***E*$_{21s}$**, ***E*$_{22s}$** ) in Figure 5.3 represent external events that stimulate responses from the system. The output events (***E*$_{3s1}$**, ***E*$_{4s1}$**, ***E*$_{1s2}$**, ***E*$_{3s2}$**) represent the externally observable behavior of the system.

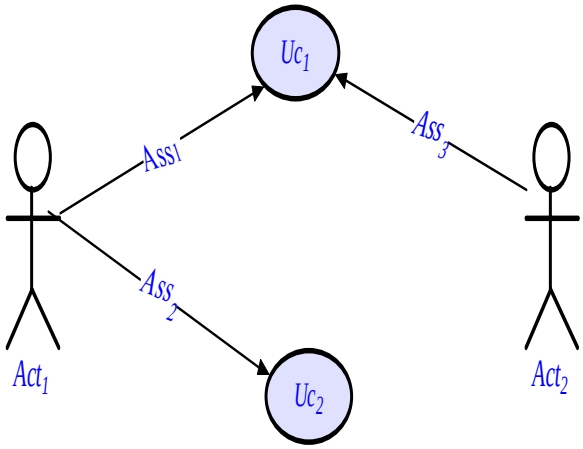



Figure 5.2 A Use Case diagram for a system *S*

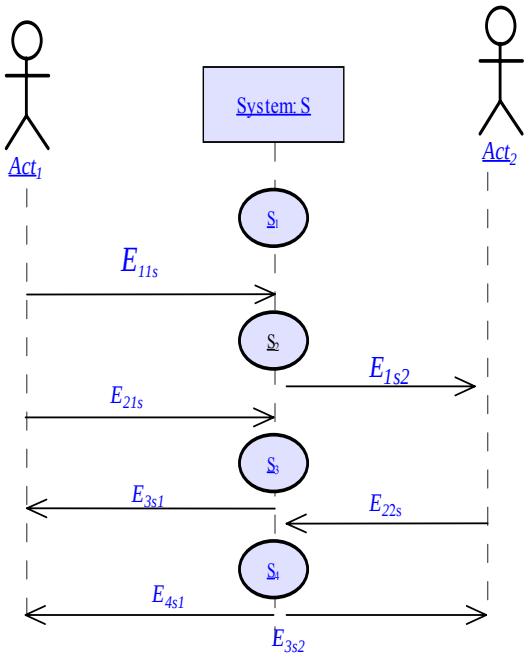



Figure 5.3 high level system sequence diagram

The process starts with FFA based on the annotated system scenario diagram Figure 5.4. We perform FFA using guide words defined in Table 5-1 [McDermid, 2000] to identify possible failure modes for each event between the system and the actors (***E*$_{11s}$**,

$E_{21s}$, $E_{22s..}$,….), and ($Act_1$, $Act_2$ ). The events are systematically examined for potential hazards which include the loss of event, the unintended delivery of event and event malfunctions. The analysis considers each event in turn and decides whether or not these *hypothetical* failure modes are credible and, if they are, what the consequences might be. This gives a clear view of how the failure of these events could contribute to hazards and accidents during the scenario. The input to FFA is a list of events of system level scenario belonging to $Uc_1$, list of guide words [McDermid, 2000], and cost of failure for every class of failure figure 5.4.

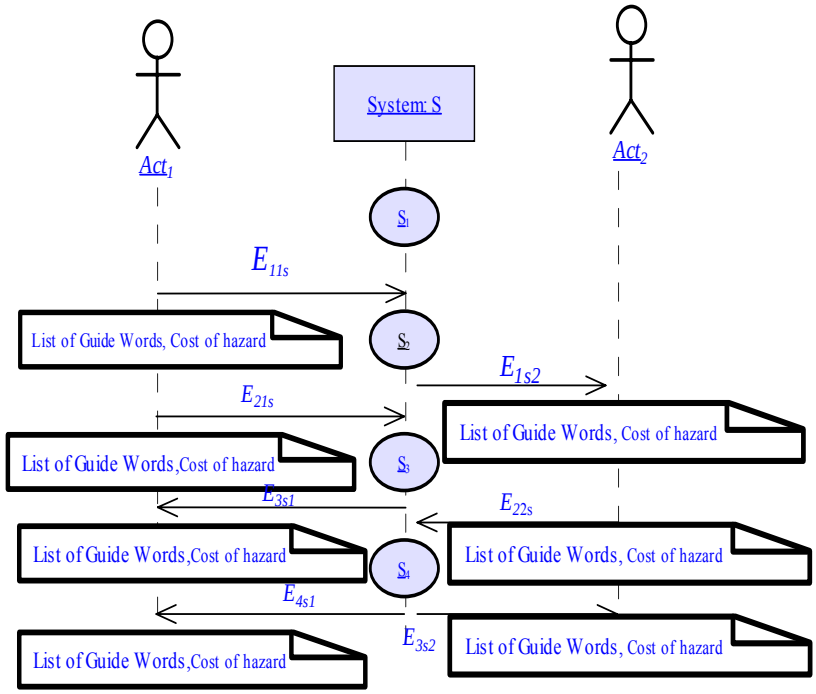



Figure 5.4 The annotated system level sequence diagram of use case ***UC1*** for system ***S***

The output of FFA is a tabulated form (see Table 5-2). The results of FFA are provided early in the design process; these results provide a comprehensive picture of the ways that the system could fail.

| Guide Word | Meaning |
|---|---|
| *Omission* | A necessary action does not occur. |
| *Commission* | An unwanted action is performed. |
| *Early* | An action is performed before the time (either real time, or relative to some other action) at which it is required. |
| *Late* | An action is performed after the time at which it is required. |
| *Value* | The timing of the action is correct, but the data with which it is performed with or upon is incorrect. |

Table 5-1 Guide words

### 5.2.2 Components/Connectors Failure Modes

FMEA examines component/connector failures considering component/connector malfunctions.

It generates a failure model for the components/connectors under examination; FMEA is essentially a tabular process. FMEA is applied for each component/connector within the sequence diagram. During specific scenario, components interact with each other by exchanging messages. Each of these interactions links a component that requests an operation with a component that performs the operation. All these interactions, collaboration and component behavior are captured in sequence diagrams. For example Figure 5.5 shows sequence diagram which corresponds to a system scenario in Figure 5.3.

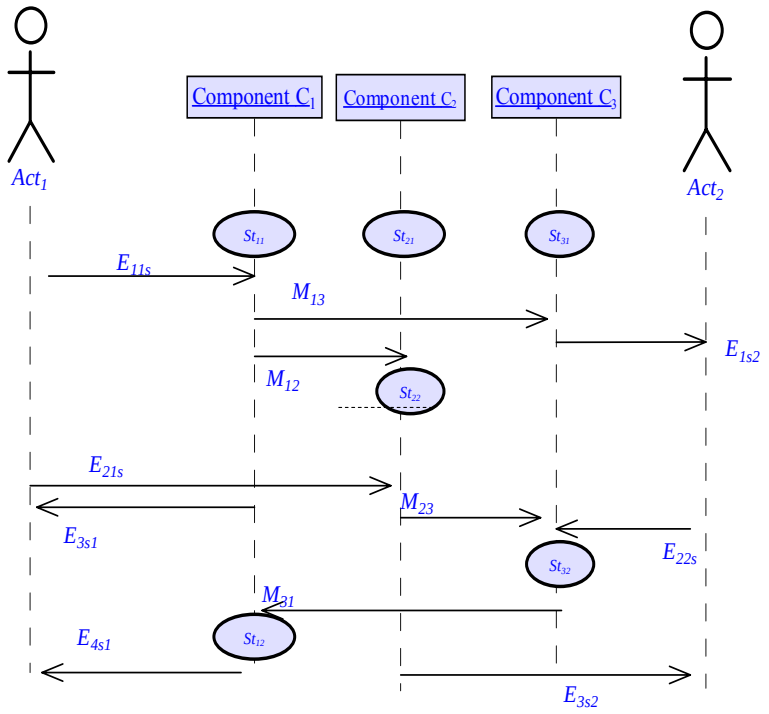



Figure 5.5 Sequence diagram of components **$C_1$**, **$C_2$** and **$C_3$** interaction

The "initiating" actor ***$Act_1$*** starts the scenario by sending the initial event to the system. The sequence diagram models messages (***$M_{12}$***, ***$M_{21}$*** , ***$M_{13}$***...) sequence among components **$C_1$**, **$C_2$**, **$C_3$**, and actors ***$Act_1$***, ***$Act_2$***.

The behavior of each component could be captured with the component state diagram during this scenario. The component changes its states through interactions based on message exchange. A hazard occurs from unwanted interactions (or events). Each of the unwanted events in the sequence is either due to a message being sent incorrectly by the sender component, or the message not being acted on correctly by the receiver component or the connector not acting well. These events can be generated by sender or receiver state transitions. Therefore faults in component state or transitions can give reasons for a component/connector failure [Lindsay, 1997]. It is necessary to

confirm that under correct behavior of components/connectors, the system doesn't allow the occurrence of the hazards. That is, if the components in the system correctly generate the intended messages, are in the correct state, and connector transmit correct message, then the system will not fail. This means that no failure will happen to components/connector.

In order to identify possible faulty behaviors for the components we can apply FMEA to the states of the components [DiMarco, 1995], [Charles, 1997]. We identify hazards associated with each component, detail all possible failure modes, and identify their resulting effect on the system. The output of this process is a tabulated form (see Table 5-3 section 5.3.2).

A Connector is defined as the interface between two components [Yacoub, 2003]. The Connector transmits the messages between the components. By applying FMEA on connectors using the messages transmitted through these connectors, we can identify connectors' failure modes and the effect of these failures on the system. We identify hazards associated with each connector, detail all possible failure modes, and identify their resulting effect on the system.

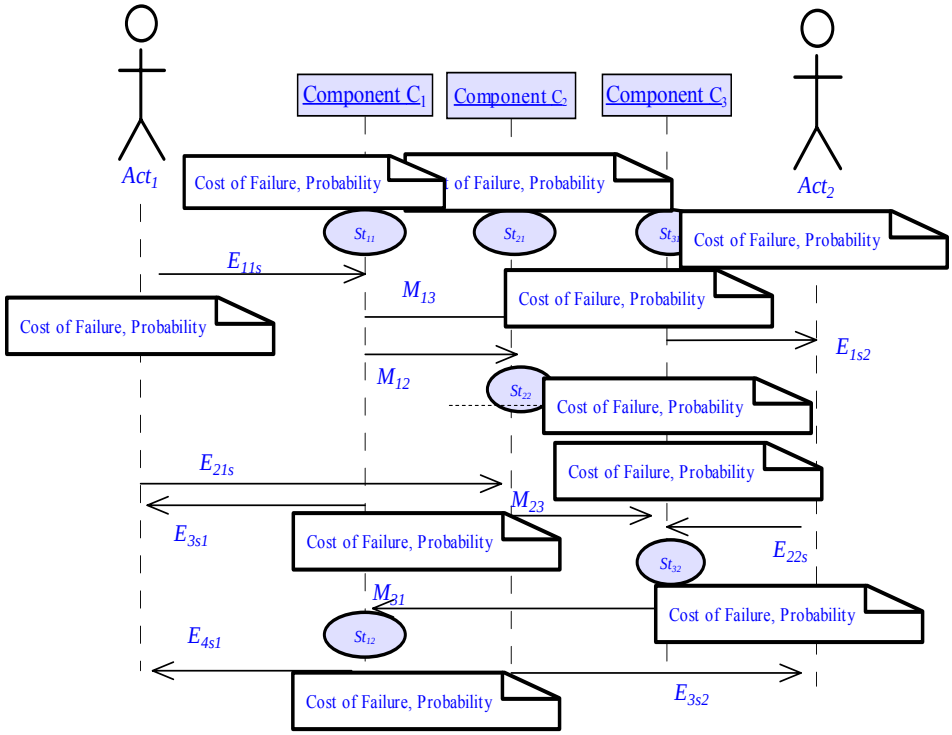



Figure 5.6 Annotated sequence diagram

### 5.2.3 Fault Tree Analysis

FTA is a top-down method used to identify failure causes [Kirsten, 1998]. FTA is primarily a means for analyzing causes of hazards, not identifying hazards. The process of analyzing causes is documented in one or more fault trees. FTA is a depiction of the logical interrelationships of the basic events that may lead to a particular undesired event. FTA is used for addressing low level failure conditions (basic events) and their potential effect for causing the top level hazards (top events) [Johannessen, 2001]. Failure of components/connectors (low level) will propagate to the system level (higher level). We use FTA to map system level hazards (output from FFA) to components/connectors failure modes (output from FMEA). The top events of the fault trees are the system level hazards and the basic events are the components/connectors failure modes.

### 5.2.4 Cost of Failure Graph

Kmenta in [Kmenta, 2000] describes failure scenarios as "*undesired cause-effect chains of events, from the initiating cause to end effect, including all intermediate effects*". Each failure scenario happens with some probability and results in negative consequences. With FTA considered as a cause effect model [Liu, 2000], using FTA results in many cause effect chains with probabilities for each cause and effect. These cause-effect chains relate the system level hazard identified from FFA to components/connectors failure modes identified from FMEA. Considering these cause-effect chains as failure scenarios for the system, we could estimate the cost of failure of each component/connector based on these failure scenarios.

Cost is an accepted measure of consequences [Gilchrist, 1993]. Expected cost is used extensively in the fields of Risk Analysis, Economics, Insurance, and Decision Theory [Gilchrist, 1993]. Kmenta and Ishii [Kmenta, 2000] proposed an adaptation of FMEA considering the consequences of the failures in terms of costs. Cost is a universal language understood by engineers without ambiguity. We develop cost of failure graph for every component/connector and scenario to estimate cost of failure of every component/connector and scenario. For a specific component/connector there is more than one failure scenario, the expected cost of failure for component/connector is the sum of all costs over these scenarios weighted by the probability of each failure scenario.

We develop a component/connector cost of failure graph [Sherer, 1988] to estimate the component/connector and scenario cost of failure using annotated UML sequence diagrams representing the interactions of components and using FFA and FTA analysis.

During the execution of a system scenario $S_x$, there are many hazards. These hazards and their consequences are identified in step 1 using FFA technique. In step 3 we estimate the probability of each of these hazards. The expected cost of failure of a system scenario may be estimated by summing the expected uses of the scenario, weighted by the expected consequences of all hazards that may result from these uses. Using the probability of usage of the scenario [Katerina, 2003], probability, and cost of these hazards for this scenario (results from step 1 and step 3) we could estimate the cost of failure of this scenario using the cost of failure graph as shown in step 4 of the methodology.

Definitions:

$^{x}Cost_i(M)$ : The cost of failure of (component/connector) $i$ in a given failure mode $M$ in a given system scenario $S_x$.

$^{x}p_i(M)$ : The probability of (component/connector) $i$ being in failure mode $M$ in a given system scenario $S_x$.

$^{x}p(H)$ : The probability of system level hazard $H$ for a given system scenario $S_x$.

$^{x}Cost(H)$ : Cost of failure for a given system hazard $H$ in a given scenario $S_x$

Total expected cost of failure of (component/connector) $i$ in a given system scenario $S_x$ is as follows:

$$^{x}TotalCost_i = \sum_{k=1}^{k=H} {}^{x}p(k) \sum_{j=1}^{j=M} {}^{x}p_i(j) \ * \ {}^{x}Cost_i(j) \qquad (1)$$

$^{x}p(S)$ :Probability of execution of a given scenario $S_x$ [Katerina, 2003]

The total expected cost of failure of a given scenario $S_x$ is estimated as follows:

$$^{x}TotalCost(S) = {}^{x}p(S) \sum_{k=1}^{k=H} {}^{x}p(k) * {}^{x}Cost(k) \qquad (2)$$

### 5.2.5 System scenario and components/connectors severity

The cost of failure metrics is a measure of consequences [Kmenta, 2000], and basing on this the cost of failure could be mapped on a 0.1-1.0 severity rank (cost severity graph) Figure 5.7.

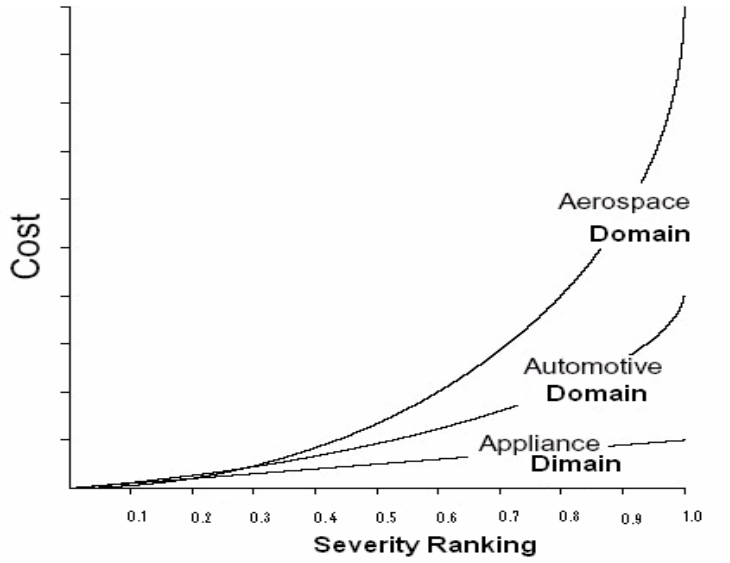



Figure 5.7 Cost-severity graph

Using cost-severity graph, we map the expected cost of failure of component/connector as well as system scenario on a severity rank as shown in Figure 5.7 cost-severity graph. The cost-severity curve depends on the application domain. For example in Health care, the cost of out-patient care would have a severity rank between 0.1-0.3, whereas in-patient care would have a severity rank between 0.3-0.6, followed by the cost-severity of intensive care. In general, the cost-severity relationship is nonlinear.

## 5.3 Case Study

We have selected a case study of a cardiac pacemaker. This case study is described in chapter 4.

### 5.3.1 FFA analysis

Figure 5.8 shows system level scenario diagram for the *AVI* mode of operation. The system received *Programmin_Command* event from the *programmer* actor to operate in *AVI* mode. To monitor the heart, the system receives *VSense* event from the heart actor and handles it. The system begins pacing the heart by sending signals which are *Pace* event to the heart actor. Using the FFA with guide words as explained in section 2.1 we come up with the FFA result in Table 5-2.

| Event Name | Class of failure | Failure | Effects on System | Cost of failure($) | Comments |
|---|---|---|---|---|---|
| *Programming_Command* | *Value* | A Fault in processing command routine | Heart is continuously triggered but device is still monitored by physician, need immediate fix or disable. | 1000 | The component received the command misinterpret it |
| *VSense* | *Omission* | Timer not set correctly | No pace is generated for the heart, patient will require intensive care | 100,000 | Some component's timers does not work well |
| *Pace* | *Commission* | Pacing hardware device malfunctioning | Heart is always paced while patient condition requires only pacing the heart when no pulse is detected. Heart operation is irregular because it receives no pacing | 100,000 | Some component's sensor failed to sense the heart. |

Table 5-2 FFA for *AVI* scenario presented in Figure 5.8

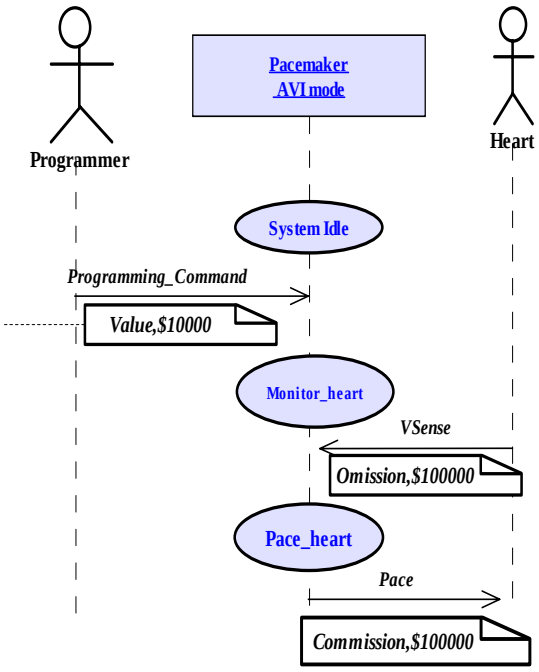



Figure 5.8 Scenario Diagram of Pacemaker System in *AVI* mode (system scenario)

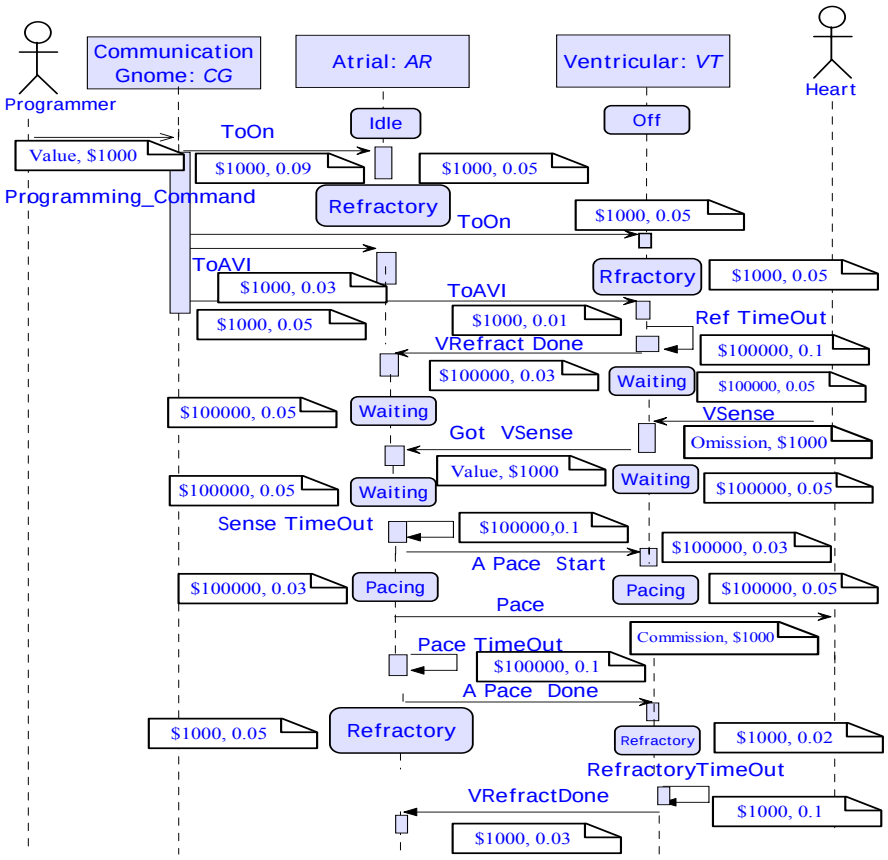



Figure 5.9 Sequence diagram of the *AVI* scenario

### 5.3.2 FMEA analysis

In the sequence diagram of Figure 5.9, the ***VT*** component monitors the heart. When a heartbeat is not sensed, the ***AR*** component paces the heart and a refractory period is then in effect. Table 5-3 is the FMEA table for ***AR*** component. Applying FMEA on every component by tracing states and transitions for every component from its state diagram, we come up with the FMEA result. Table 5-3 is the result of FMEA for component ***AR.*** Also we apply FMEA for each connector by tracing all messages transmitted over the connector. Due to space limitations, we show the results concerning the ***AR*** component only.

### 5.3.3 FTA analysis

Figure 5.10 shows a fault tree of top event "***Commission***" of "*Pace*" hazard as a function of components/connectors failure modes. FTA step 3 combines the results from FFA step 1 and FMEA step 2 to map the ***Commission*** "*Pace*" hazard to its basic failure modes.

| Component | Failure Modes | Effect on the system | Cause of failure | Cost of Failure $ |
|---|---|---|---|---|
| *AR* | *ToOn Value Error* | The component will not work and there is no pace of the heart | The component does not receive signal from CG | 1000 |
| - | *VR stuck in Refractory State* | The component will stay in Refractory state and there is no pace | Connector VT-AR sends a wrong message, or component AR failes to understand the message. | 1000 |
| - | *The component receive GotVSense but there is no pace (Stuck in Waiting state)* | The component will stay in waiting state and there is no pace | The component sensor does not work | 100000 |
| - | *Sense TimeOut Error* | The component in waiting state, heart operation is irregular because it receives no pacing | The component sensor does not work or value of Sense Time is wrong | 100000 |
| - | *PaceTimeOut Error (component stuck in Pace state)* | Heart is always paced while patient condition requires only pacing the heart when no pulse is detected | There is a problem in the sensor, or the timer does not work | 100000 |

Table 5-3 FMEA for *AR* component

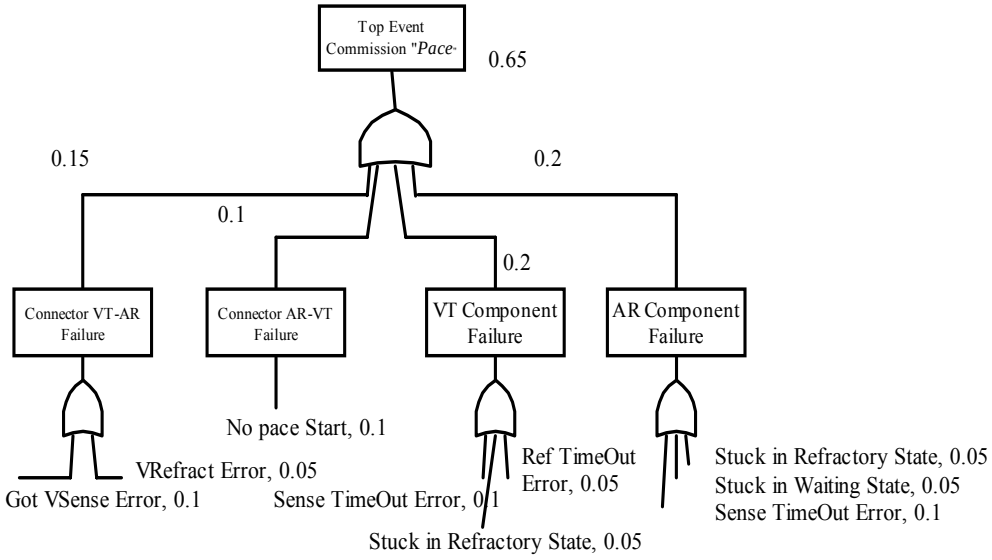



Figure 5.10 Commission "*Pace*" Fault Tree

Using the probabilities of the basic events which are determined in step 2, we estimate the probability of top level events.

### 5.3.4 Component/Connector and scenario cost of failure graph

The first level of the AR component cost of failure graph shown in Figure 5.11 is the top events of all fault trees with their probabilities. Every component/connector could contribute to these hazards during the execution of the scenario. The component/connector contribution to these hazards results from the component/connector failure modes. To estimate the cost of failure of component/connector during the scenario we develop the cost of failure graph which relates component/connector failure modes with the system level hazards. The probability of each failure mode is derived from domain knowledge and it could be annotated with the sequence diagram.

Using equation (1) we could calculate the estimated cost of failure for every component/connector. The second level of the cost of failure graph Figure 5.11 is the failure modes of the ***AR*** component associated with the system level hazards (first level

of the graph). The third level of the graph is the consequence of every failure mode of the ***AR*** component represented by cost.

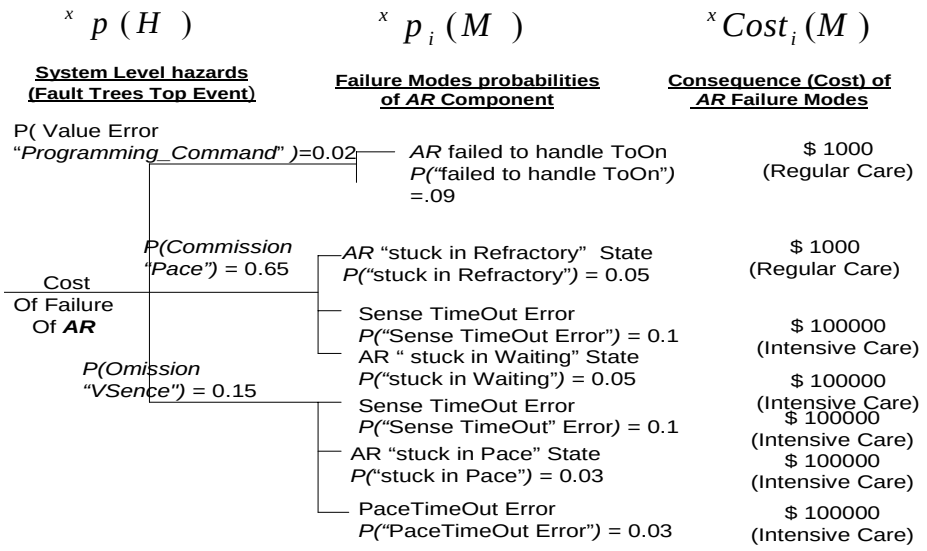



Figure 5.11 cost of failure graph of the ***AR*** component

During the intended use of the *AVI* scenario there are several system level hazards. The output of the FFA is the list of these possible hazards. Every hazard is represented by a top event in a single fault tree. As shown in Figure 5.9 the *AVI* scenario is used to initialize the system through *Programming_Command* event (*Programmer* actor programs the pacemaker to work in *AVI* mode), monitor the heart through *VScense* event (pacemaker receive signal from *Heart* actor) and pacing the heart through *Pace* event (pacemaker pace the heart). The probability of usage of *AVI* scenario is given in chapter 4. Using this probability of usage with the results from step 1 (list of system level hazards, cost of hazards) and results from step 3 (probability of the system level hazards), we could estimate the cost of failure of the scenario. To implement this, we use the cost of failure graph in Figure 5.12 and equation (2). Based on equation (2) the estimated total cost of failure for the *AVI* scenario is 23205.8 $, similarly based on equation (1) the estimated total cost of failure of AR component is 12,184.3 $.

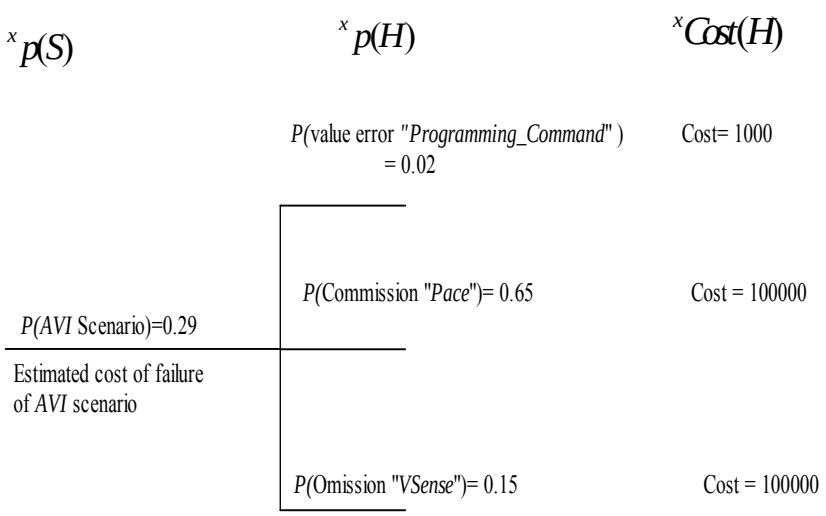



Figure 5.12 Cost of failure graph of *AVI* scenario

### 5.3.5 Components/Connectors and Scenario severity

In the final step of the methodology we use a cost severity-graph (Figure 5.13) to determine the severity rank for each component/connector as well as the scenario. For the *AVI* scenario this is done by extending point A in y axis which gives the total cost of failure of the scenario, to meet the Cost-Severity curve at point B. We extend point B to meet the severity scale in the x axis at point C. Point C gives the severity value associated with the scenario failure. Table 5-4 shows the results of the final step of the methodology after mapping the cost of failure of each component/connector to severity rank.

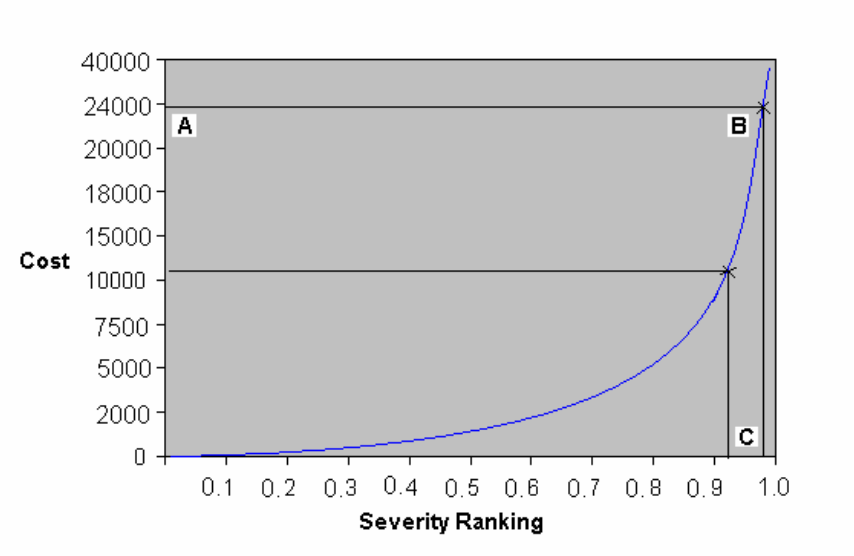



Figure 5.13 cost-severity graph

| Component/Connector Name | Severity |
|---|---|
| **Connector CG-AR** | 0.50 |
| **Connector CG-VT** | 0.50 |
| **Connector AR-VT** | 0.94 |
| **Connector VT-AR** | 0.95 |
| **Component CG** | 0.50 |
| **Component AR** | 0.96 |
| **Component VT** | 0.95 |

Table 5-4 Severity of each components/connectors in *AVI* scenario

Next we map the estimated cost of failure of *AVI* scenario according to severity rank using a cost-severity graph. Table 5-5 shows the severity of AVI scenario.

| Scenario | Severity |
|---|---|
| AVI | 0.96 |

Table 5-5 *AVI* scenario severity

The results from Table 5-4 show that the VT and AR components are components with the highest severity rank in the ***AVI*** scenario. This result is intuitive since these two components are the most active and the most critical components that directly control the operation of the heart during the scenario. The CG component, on the other hand, controls the programming operation, and is monitored by the physician before the device is put into operation.

Also from Table 5-4, we identify that the connection between the VT and AR components (*AR-VT*, *VT-AR* connectors) are the highest severity connectors. This result is also intuitive in the context of the pacemaker example, since these connectors deliver critical messages controlling the heart operation such as sensing and pacing.

Results from Table 5-5 shows that the *AVI* scenario is a high severity scenario because it controls the pacing operation of the heart; the worst consequence of failure of this scenario could lead to patient's death. Combining these results with the estimated probability of failure developed in [Katerina, 2003] for software architectural element (component/connector) as well as system scenario, we have developed an architectural level risk assessment methodology [Katerina, 2003]. Using the risk methodology, we could estimate the risk factor for every component/connector, scenario, use case and the whole software system. This architectural level risk assessment is explored in detail in [Katerina, 2003].

We have used this methodology to estimate the severity for severity assessment in the performance risk assessment process measured in [Cortellessa, 2004]. This proposed methodology for severity assessment is used to assess the severity of software system

scenarios with requirement failure modes to estimate architectural level requirement risk assessment [Appukutty, 2005].

## 5.4 Conclusions and future work

In summary, this research describes a methodology for estimating severity of each software architectural element (component/connector) at the software architectural level as well as severity of system scenarios. The methodology is based on dynamic UML specifications, taking into account the possibility of component/connector cost of failures. This methodology incorporates the cost of failure to severity rank mapping. FFA is used as a top down approach based on system scenarios to identify the system level failures, FMEA is used as a bottom up appraoch based on the detailed view of the system to identify the possible causes component/connector failures, and FTA correlates the results of FMEA and FFA. By annotating the hazard analysis results and the cost of failure information in the UML diagrams, this methodology of estimating severity can be automated in development environments supporting UML. We used the proposed methodology for the severity in estimating the performance-based and requirement-based risk factor of software systems [Appukutty, 2005] [Cortellessa, 2004].

In the future work we will apply this methodology to bigger NASA case studies to study the validity of this methodology. In the future work we would develop a tool support which will help domain expert to apply this methodology.

# *Chapter 6*

## Risk Assessment

### 6.1 Introduction

In this chapter, we explore the proposed architectural level risk assessment methodology for assessing risk of architectural element component/connector risk. The basis for the proposed risk assessment methodology is the use case and scenario diagrams of the system UML model. The use cases and scenarios of a UML specification model drive this methodology. We assume that the UML logical architectural model consists of a use case diagram defining several independent use cases (in future work, we will explore the dependent use cases), and that each use case is realized with one or more independent scenarios modeled using sequence diagrams. The proposed methodology identifies the potential risks in the software architecture, based on the early system specifications. The architectural specifications are the UML models that are available early in the software life cycle.

Based on the risk definition [NASA3, 2000 the probability of software failure is explored in chapter 4.

As shown in chapter 4 the estimation of the probability of failure of software component/connector is defined as the normalized dynamic metrics. Component probability of failure is the normalized dynamic complexity and the connector probability of failure is the normalized dynamic coupling [Katrina, 2003], [Hassan, 2001]. In chapter 5 we showed how to assess the severity of failure for each component/connector in the software system [Hassan1, 2003], [Hassan2, 2003], [Hassan, 2005]. Combining the two

factors (probability of failure, severity of failure) we show in this chapter how we can estimate risk for software component/connector, scenarios, use cases and system. The estimation of these risk factors is based on the proposed risk assessment methodology.

The proposed risk analysis methodology iterates on the use cases and the scenarios that realize each use case and determines the component/connector risk factors for each scenario, as well as the scenarios and use cases risk factors. For each scenario, the component/connector risk factors are estimated as a product of the dynamic complexity behavioral specification measured from the UML sequence diagrams (see chapter 4) and the severity level is estimated using hazard analysis and cost of failure (see chapter 5). Then, a Markov model [Ajith, 2004] is constructed for each scenario to determine a scenario risk factor. Further, the use cases and overall system risk factors are estimated. The outcome of this process is a list of critical scenarios in each use case, a list of critical use cases, and a list of critical components/connectors for each scenario and each use case. Also we estimate the system level risk factor.

We proceed in this chapter as follows: Section 6.2 is the proposed risk assessment methodology, Section 6.3 component/connector risk factor, Section 6.4 describes the scenario level risk factor model, use case and system level risk factor are described in Section 6.5, Section 6.6 is the case study, sensitivity analysis is explored in Section 6.7 and the chapter summary at Section 6.8.

**6.2 Risk Assessment Methodology**

The proposed methodology is modeled using UML as shown in figure 6.1. Figure 6.1 describes the use case model of the risk assessment methodology. To automate this risk assessment methodology the UML model is implemented as a prototype tool and is

described in chapter 8 [Wang, 2003]. The use case model of the proposed methodology is described as follows:

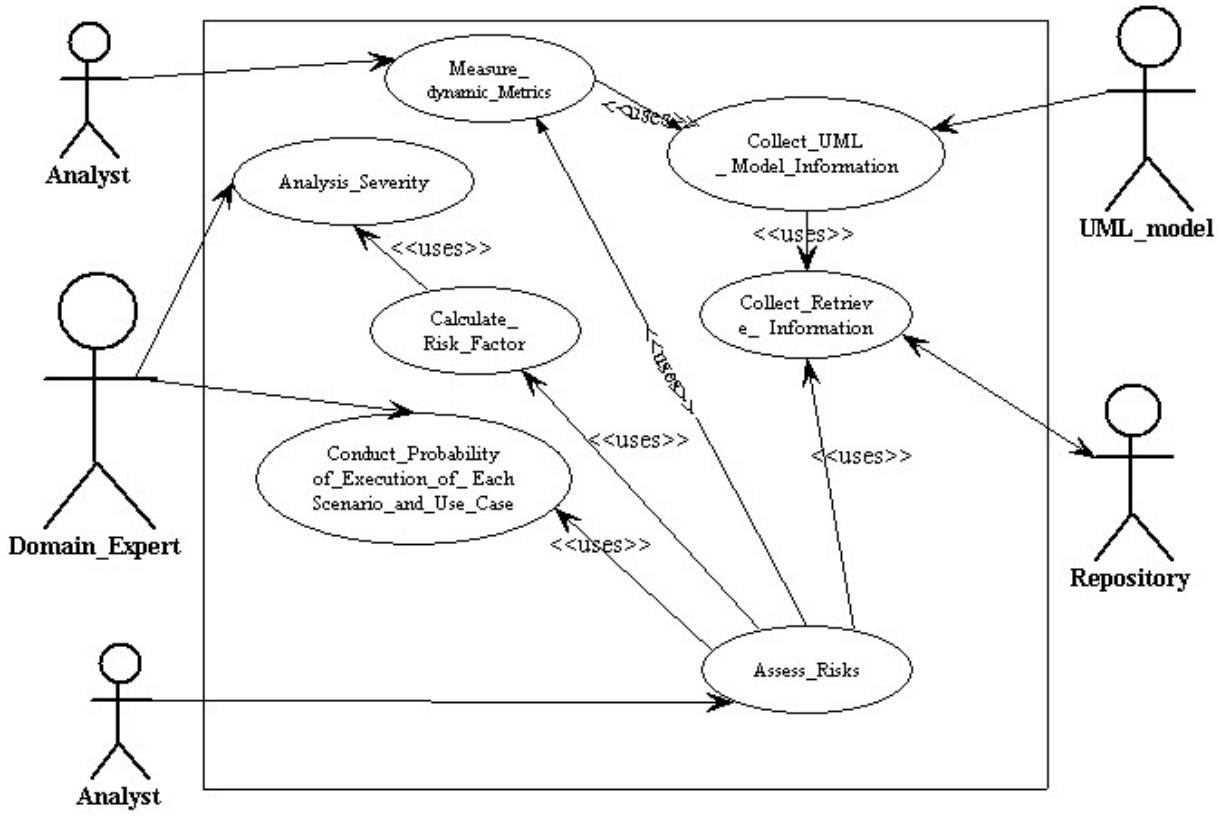



Figure 6.1 Overall Use Case UML model of the proposed methodology

The whole process begins with step 1 which consists of collecting information from UML visual model; this step is explored in details in chapter 8 and the process continues as described below. Step 2 is explored in chapter 4, step 3 is explored in chapter 5, and the other steps are explored in this chapter.

1. Collect the information of the software system from UML model (see chapter 8) [Wang, 2003],
2. Estimate dynamic metrics [Hassan, 2001] for each component and connecter of the software system for a given scenario in a given use case (see chapter 4),

3. Conduct severity analysis for the software system for a given scenario in a given use case (see chapter 5) [Hassan1, 2003], [Hassan2, 2003] [Hassan, 2005],
4. Calculate a heuristic risk factor [Ammar, 1997] for each component and connecter in the software architecture for a given scenario in a given use case (see section 6.3),
5. Conduct risk factor for each scenario in a given use case based on component/connector risk factor (see section 6.4) using Markov model [Katerina, 2003].
6. Conduct risk factor for a each use case in the system based on scenario risk factor and conduct risk factor for the overall system based on each use case risk factor (see section 6.5),

The core of the proposed methodology [Katerina, 2003] is defined by the following algorithm steps:

*For each use case*
*For each scenario*
*For each component*
*Measure dynamic complexity*
*Assign severity based on cost of failure and hazard analysis*
*Calculate component's risk factor*
*For each connector*
*Measure dynamic coupling*
*Assign severity based on hazard analysis*
*Calculate connector's risk factor*
*Generate critical component/connector list*
*Construct Markov model & Calculate transition probabilities*
*Calculate scenario's risk factor*
*Rank the scenarios based on risk factors, Determine critical scenarios list*
*Calculate use case risk factor*

*Rank use cases based on risk factors, Determine critical use case list*
*Calculate overall system risk factor*

The methodology is a top down approach and it is iterative over each level of the software architecture, starting from use case level, to scenario level and down to basic component/connector level. The process starts from a use case level, iterates through each use case and each scenario of that use case. From the scenario level, each component and connector is analyzed and the corresponding risk factors are estimated. The component/connector risk factor is the product of the normalized dynamic metrics (see chapter 4) and the severity of the failure of that component/connector (see chapter 5). The scenario risk factor is estimated based on the Markov model which is described in [Ajith, 2004]. Use case and system level risk factors are estimated as shown in section 6.5 taking into consideration the assumption that use cases are independent.

### 6.3 Component/Connector Risk Factor

Components and connectors are the building blocks of any software architecture. Hence the architectural risk factor is dependent on the risk factors of the components/connectors. The assessment of component/connector risk factors is based on the dynamic UML specifications; these risk factors are referred to as the dynamic heuristic risk factors [Katerina, 2003], [Ammar, 1997].

The risk factor of component/connector is the product of the probability of failure of each component/connector and the severity of these failures. The probability of failure of component/connector is estimated based on dynamic UML specifications (see chapter 4), and the severity of failure is estimated based on hazard analysis techniques and cost of that failure (see chapter 5).

### 6.3.1 Component Risk Factor

For each scenario $S_x$, we calculate heuristic risk factors for each component participating in the scenario based on the dynamic complexity and severity level. Note that in general, these values will be different for different scenarios. The risk factor $rf_i^x$ of a component $i$ in scenario $S_x$ is defined as

$$rf_i^x = DOC_i^x * svt_i^x \qquad 6.1$$

where $DOC_i^x$ $(0 \leq DOC_i^x \leq 1)$ is the normalized complexity of the $i^{th}$ component in the scenario $S_x$ (chapter 4), and $svt_i^x$ $(0 \leq svt_i^x < 1)$ is the severity level for the $i^{th}$ component in the scenario $S_x$ (chapter 5).

### 6.3.2 Connector Risk Factor

The risk factor $rf_{ij}^x$ for a connector between components $i$ and $j$ in the scenario $S_x$ is given by

$$rf_{ij}^x = EOC_{ij}^x \cdot svt_{ij}^x \qquad 6.2$$

where $EOC_{ij}^x$ $(0 \leq EOC_{ij}^x \leq 1)$ is the normalized coupling for the connector between $i^{th}$ and $j^{th}$ components in the scenario $S_x$ (chapter 4), and $svt_{ij}^x$ $(0 \leq svt_{ij}^x < 1)$ is the severity level for the connector between the $i^{th}$ and the $j^{th}$ components in the scenario $S_x$ (chapter 5).

## 6.4 Scenarios Risk Factor

*This step of the methodology has been presented here for the sake of completeness and the detailed implementation is presented in [Ajith, 2004]*

We use an analytical modeling approach to derive the risk factor of each scenario. For this purpose, we generalize the state-based modeling approach previously used for architecture-based software reliability estimation [Katerina, 2001]. Thus, the software reliability model first published in [Cheung, 1980] considers only component failures. In the scenario risk model, we account for both component and connector failures, that is, we consider both component and connector risk factors. In addition, instead of a single failure state for the scenario, we consider multiple failure states that represent failure modes with different severity. This approach allows us to derive not only the overall scenario risk factor, but also its distribution over different severity classes, which provide additional insights important for risk analysis.

For example, the two scenarios may have close values of scenario risk factor with significantly different distributions among severity classes. It can then be inferred that the scenario with a risk factor distributed among more severe failure classes (e.g., critical and catastrophic) deserves more attention than the other scenario.

The scenario risk model is developed in two steps. The first step is to build a control flow graph, which is a direct translation of the scenario diagram. This control flow graph is constructed using the UML sequence diagrams and describes software execution behavior with respect to the manner in which different components interact to achieve the scenario mission. It is assumed that a control flow graph has a single entry ($S$) and a single exit node ($T$) representing the beginning and the termination of the execution, respectively.

The second step is to build the scenario risk model for that control flow graph, which is based on the DTMC. The derivation of the DTMC from the control flow graph

using the UML sequence diagram is the main step in deriving the k-step transition probability for the absorbing states steady states of the reliability model (for each sequence diagram) [Ajith, 2004]. The states in the control flow graph represent active components, while the arcs represent the transfer of control between components (i.e. connectors). It is further assumed that the transfer of control between components has a Markov property meaning that, given the knowledge of the component in control at any given time, the future behavior of the system is conditionally independent of the past behavior. This assumption allows us to model software execution behavior for scenario $S_x$ with an absorbing Discrete Time Markov Chain (DTMC) with a transition probability matrix

$$P^x = [P_{ij}^x], \qquad 6.3$$

where $P_{ij}^x$ is interpreted as the conditional probability that the program will next execute component $j$, given that it has just completed the execution of the component $i$. After building the risk model, we solve the Markov chain to estimate scenario risk factor and risk factor distribution of this scenario.

### 6.5 Use Cases and Overall System Risk Factors

The risk factor $rf_k$ of each use case $U_k$ is obtained by averaging the risk factors of all scenarios $S_x$ that are defined for that use case,

$$rf_k = \sum_{\forall S_x \in U_k} rf^x * P_k^x \qquad 6.4$$

where $rf^x$ is the risk factor of scenario $S_x$ in use case $U_k$ and $p_k^x$ is the probability of occurrence of scenario $S_x$ in the use case $U_k$.

Similarly, the overall system risk factor is obtained by averaging the use case risk factors

$$rf = \sum_{\forall U_k} rf_k * P_k \qquad 6.5$$

where $rf_k$ and $p_k$ are the risk factor and probability of occurrence of the use case $U_k$. It is obvious from equations (4) and (5) that the use cases and overall system risk factors depend on the probabilities of scenarios occurrence $P_k^x$ in the use case $U_k$ and the probability of use case occurrence $P_k$. Hence, scenarios (use cases) with high risk factors but very low probability of occurrence will not contribute significantly to the overall system risk factor.

**6.6 Case study**

We have selected a case study of a cardiac pacemaker device to illustrate how our proposed methodology works. A cardiac pacemaker is an implanted device that assists cardiac functions when the underlying pathologies lower intrinsic heartbeats low [Douglass, 1998]. An error in the software operation of the device can cause loss of a patient's life. This is an example of a critical real-time application. We use the UML real-time notion to model the pacemaker. The detailed description and UML model of pacemaker is obtained in chapter 7.

**6.6.1 Components and Connector Risk Factor**

For this case study, only one scenario was available for each use case. However, the proposed methodology is more general and supports multiple scenarios defined for each use case. We will also apply the methodology on HCS case study (see chapter 7) which supports multiple scenarios for each use case.

Figure 6.2 shows the sequence diagram of a programming scenario. In this scenario, the programmer interacts with the *RS* and *CD* components to input a set of 8 bits specifying an operation mode for the pacemaker. This byte is received by the *CG* component, which, in turn, sets the operation mode of the *AR* and *VT* components to one of five modes of operation. The other scenarios are shown in chapter 7. Using equations 6.1, 6.2 we could estimate the component/connector risk factors.

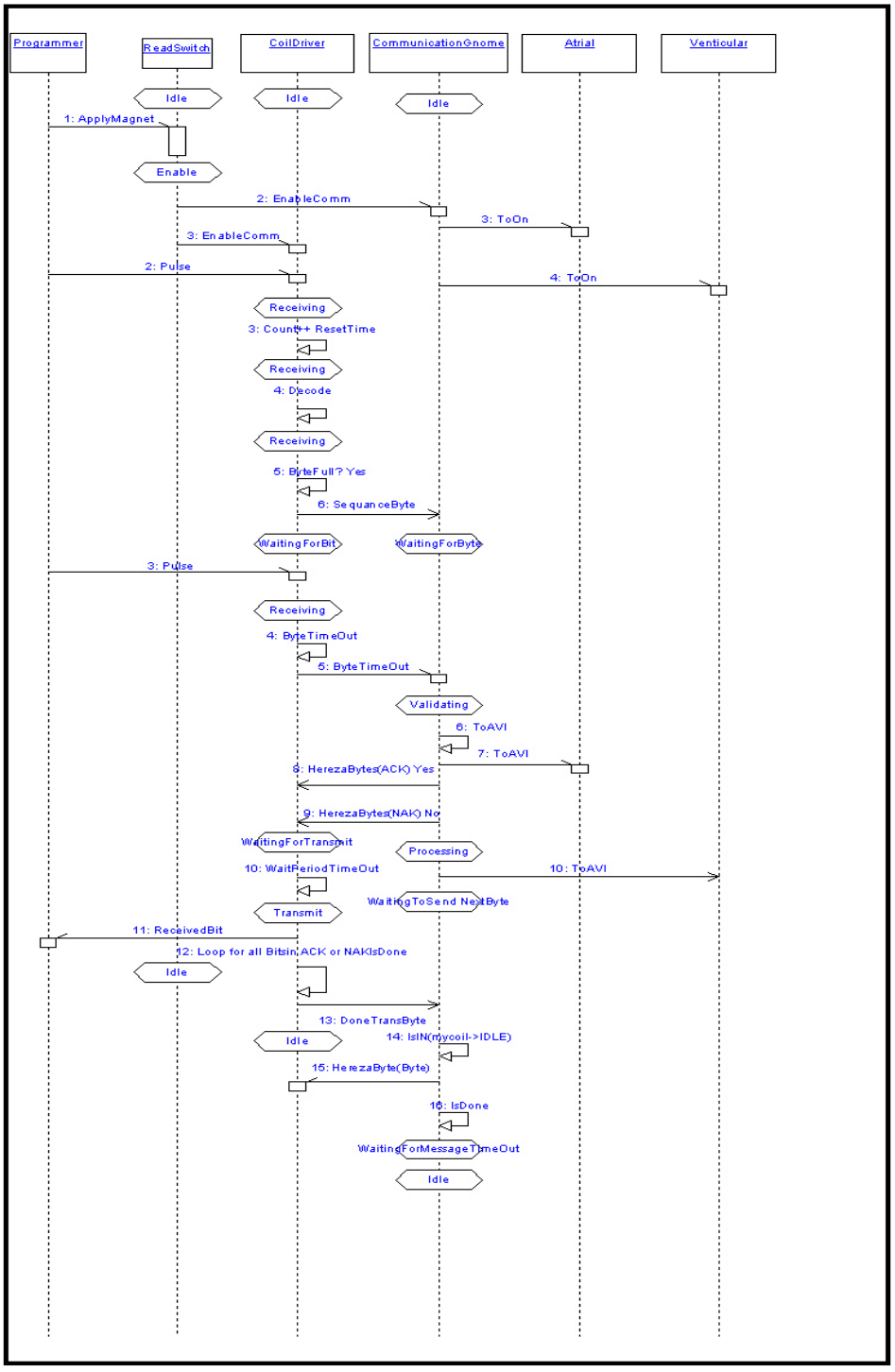



Figure 6.2 Sequence diagram of the programming scenario

A beneficial outcome of our risk assessment methodology is the ability to identify a set of most critical components. Figure 6.1, 6.2 presents risk factors of all components/connectors for different scenarios of the pacemaker case study. In these figures, the different severity levels are presented by different shades. It is obvious that *VT* and *AR* are the most critical components in the pacemaker case study since they have high risk factors with catastrophic severity in multiple scenarios. A similar approach can be used to identify the set of most critical connectors. It is also obvious from Figure 6.4 that connector *VT-AR* is the most critical connector in this case study since it has a high risk factor with catastrophic severity.

| | | Component risk factor | | | | |
|---|---|---|---|---|---|---|
| | | RS | CD | CG | AR | VT |
| Scenario Name | *Programming* | 0.05 | 0.125 | 0.15 | 0 | 0 |
| | *AVI* | 0 | 0 | 0.00016 | 0.300675 | 0.3309515 |
| | *AAI* | 0 | 0 | 0.00045 | 0.94905 | 0 |
| | *VVI* | 0 | 0 | 0.00045 | 0 | 0.94905 |
| | *AAT* | 0 | | 0.00025 | 0.949525 | 0 |
| | *VVT* | 0 | 0 | 0.00025 | 0 | 0.949525 |

Table 6.1 components risk factor

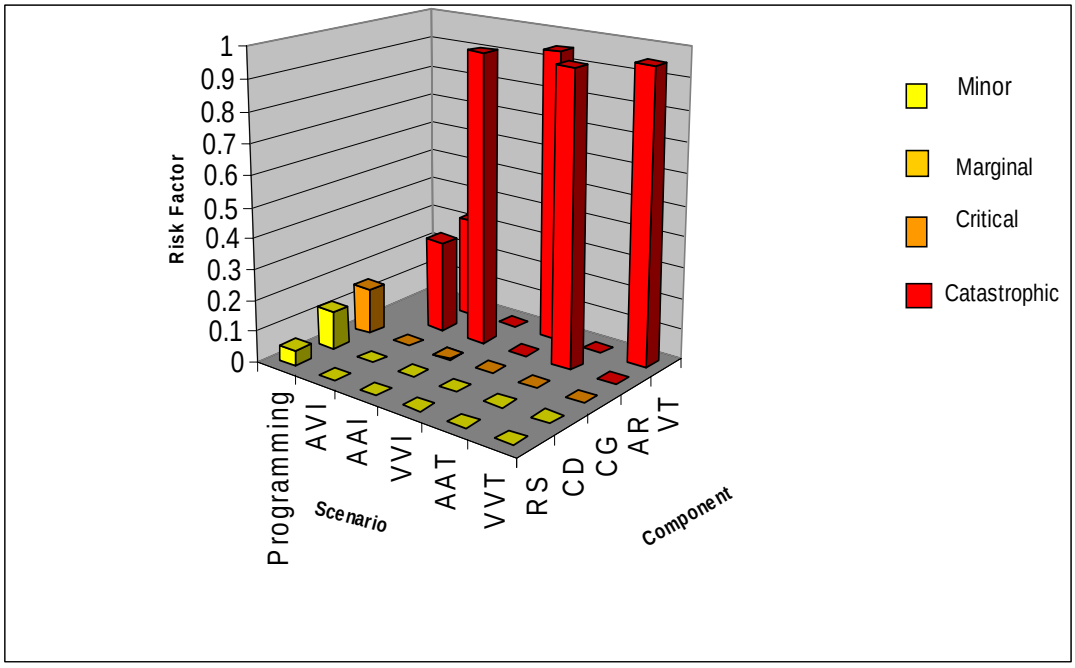



Figure 6.3 The 3-D bar graph of risk factors of the components Vs. scenarios of the cardiac pacemaker.

| | | Scenario name | | | | | |
|---|---|---|---|---|---|---|---|
| | | Programming | AVI | AAI | VVI | AAT | VVT |
| Connector Name | RS-CD | 0.03125 | | | | | |
| | RS-CG | 0.03125 | | | | | |
| | CD-CG | 0.09375 | | | | | |
| | CG-CD | 0.09375 | | | | | |
| | CG-AR | | 0.000195 | 0.5 | | 0.5 | |
| | CG-VT | | 0.000195 | | 0.5 | | 0.5 |
| | VT-AR | | 0.855 | | | | |
| | AR-VT | | 0.09215 | | | | |

Table 6.2 Connectors risk factor.

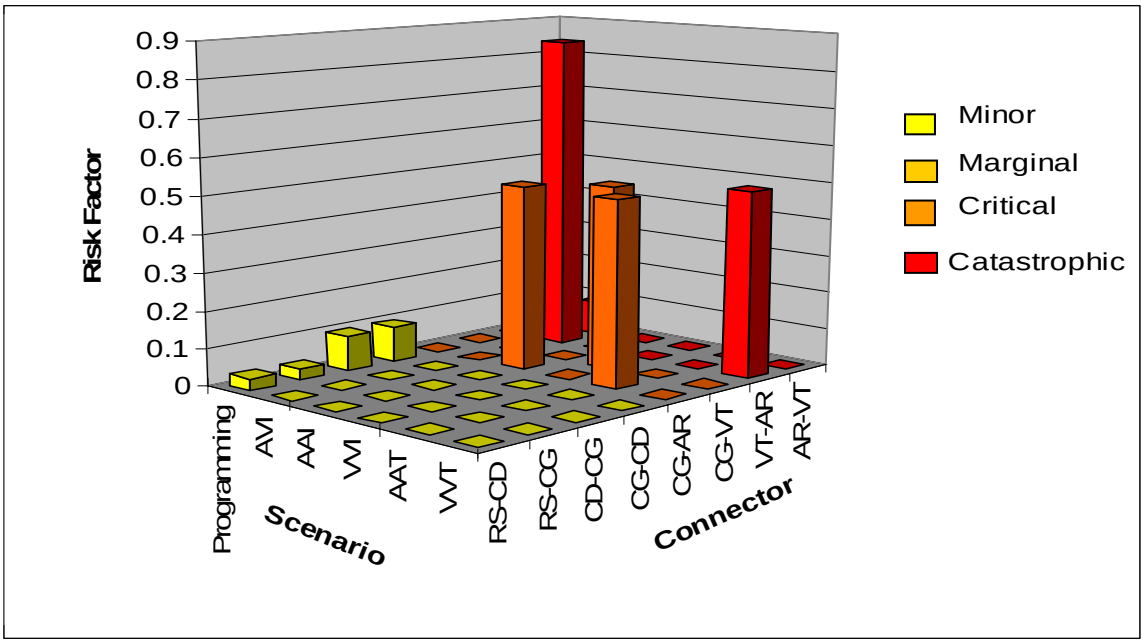



Figure 6.4 The 3-D bar graph of risk factors of the connectors vs. scenarios of the cardiac pacemaker.

Figure 6.4 shows a 3-D bar graph of the risk factors of the connectors, corresponding to their scenarios shaded according to their severity. This is one way of clearly identifying the most risky components/connectors - the component/connector with tall bars is the most risky. Another way is to rank the components/connectors according to their risk factors and severity.

From Figure 6.4, we identify that the connection between the *VT*, *AR* components are the highest risk connectors. This result is intuitively correct in the context of the pacemaker example since these connectors deliver critical messages controlling heart operation, such as sensing and pacing.

### 6.6.2 Scenario Level Risk Factor

We have developed scenario risk models for all scenarios of the pacemaker example (programming, *AVI, AAI, VVI, AAT,* and *VVT*). We will explore in some detail the *AVI* scenario. Figure 6.5 shows *AVI* scenario under the *AVI* use case. In this scenario,

the *VT* component keeps sensing the heart and the *AR* component paces the heart whenever a heart beat is not sensed. As in all scenarios, a refractory period is then in effect after every pace.

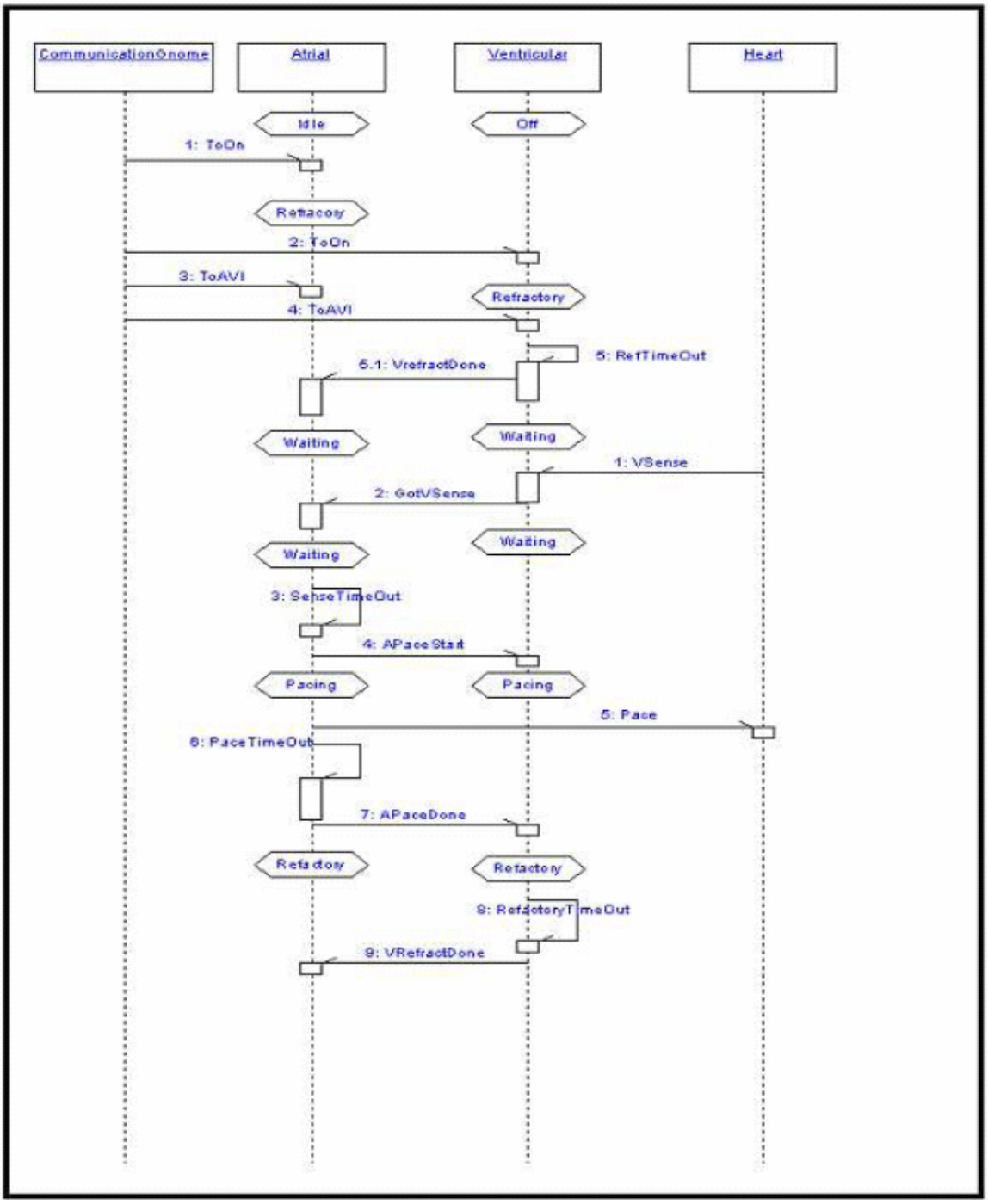



Figure 6.5 *AVI* scenario diagram

### 6.6.3 Building the Control Flow Graph from Sequence Diagram

The difference between the sequence diagram and the control flow graph is that the control flow graph has a single macro state for a component, while in the sequence diagram we have different active states of a component represented along the component's object life time. Thus the states in a control flow graph represent the active components (or, to be more precise, the corresponding active state of that component, hidden in the representation). The arcs connecting the components (i.e. connectors) represent a transfer of control between these components. Figure 6.5 shows a sequence diagram from the *AVI* scenario of a cardiac pace maker system. The sequence diagram consists of three main components - Communications Gnome (*CG*), programmed by the user to set a particular mode of system operation (in this case *AVI* mode), Atrial component (*AR*) and Ventricular component (*VT*) (which sense/pace the heart depending on the mode of operation). The heart shown in the sequence diagram is an external actor, which is sensed and paced by the pace maker system. The states of the *AR* and the *VT* components - idle, refracting, waiting, pacing are shown along the object life lines. Now this sequence diagram is converted to a control flow graph Figure 6.6. After obtaining the control flow graph, we add the probabilities for control transfers from a component to another (represented as a number along the corresponding connector). These probabilities correspond to the transition probabilities of the $P^x$ matrix. This gives the DTMC of the software execution behavior for that scenario. Figure 6.6 shows the DTMC built for the *AVI* scenario shown in Figure 6.5. It has a single entry state (state *S*), which is the dummy start state. An assumption here is that the control transfer between any of the states has the Markov property: given the knowledge of the component in control at any given time

the future behavior of the system (or in other words the next transition) is conditionally independent of the past behavior. We now assign the basic transition probabilities of the control transfer from component to component, denoted by $P^x$ (equation 3) for scenario $S_x$ [Ajith, 2004]. The matrix Figure 6.7 shows the transition probability matrix $P_{AV\,I}$ for the *AVI* scenario.

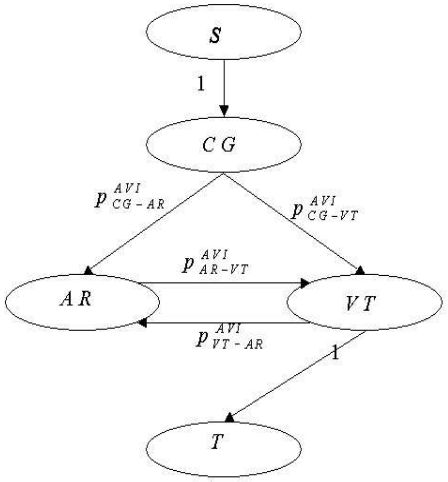



Figure 6.6: DTMC for the software execution behavior of the AVI scenario.

$$P^{AVI} = \begin{array}{c} \\ S \\ CG \\ AR \\ VT \\ T \end{array} \begin{array}{c} \begin{array}{ccccc} S & CG & AR & VT & T \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.5 & 0 & 0.5 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \end{array}$$

Figure 6.7 Transition probability metrics $I$ for the AVI scenario

### 6.6.4 Building the Risk Model

Instead of a single failure state considered in all existing architecture-based software reliability models presented in [Katerina, 2001], we consider multiple failure

states that represent failure modes with different severities. Since severity plays an important role in risk assessment, we added *4* failure states corresponding to the *4* failure modes with different severity (see chapter 5). From our severity analysis we have came up with four classes of severity. Thus, we have $n + 1$ transient states ($n$ components and the dummy start state *S*) and have five absorbing states (i.e. four failure states and one normal terminating state *T*). There could be a failure transition from a component/connector to the failure absorbing state (Figure 6.7) depending on the severity of the failure of that component/connector. If there is no failure throughout the execution of the scenario, the control reaches the normal absorbing state (the T state). The failure states in our methodology are Minor, Marginal, Critical and Catastrophic (see chapter 5).

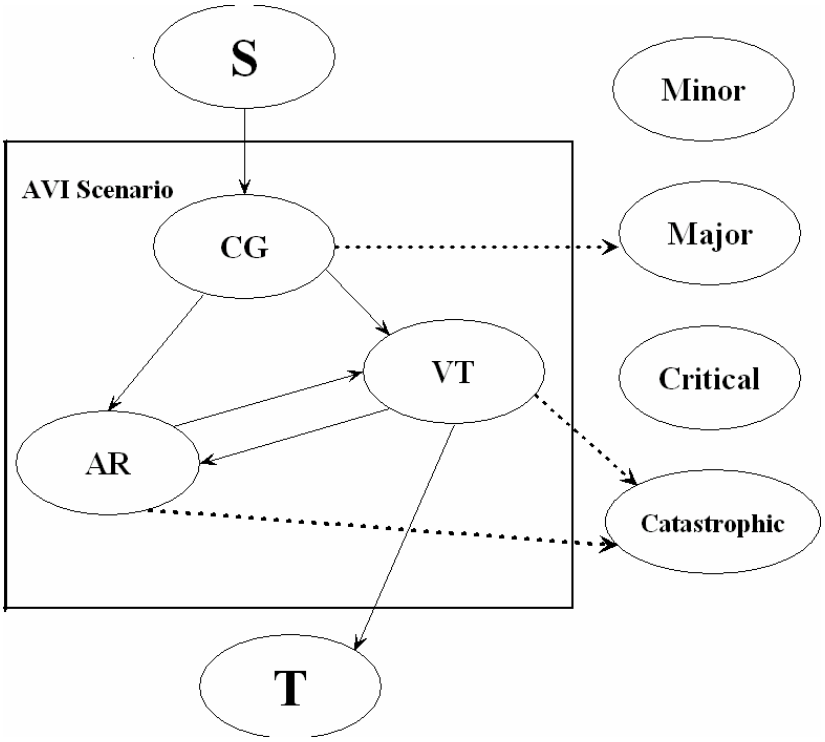


Figure 6.8 DTMC model for *AVI* scenario

### 6.6.5 Solving the Markov Chain

Dotted lines show the failure states of the scenario. Since the severity associated with component *AR* and the connector (*AR - V T)* is the same (Catastrophic), there is only

one dotted line to represent the failure transition from the *AR* representing the failures of component and connector (shown in Figure 6.8. The detailed calculation of the normal and failure transitions is presented in [Ajith, 2004]. After calculating the transition probabilities for all components and connectors we solve the risk model of the *AVI* scenario shown in Figure 6.8.

The important advantage of this risk assessment methodology is that the risk factor of the scenario is given as four factors, one for each class of severity. Since severity plays an important role in risk assessment, this concept of the severity specific risk factor provides vital meaning of the risk factor rather than a single number. Since the risk factor (0.7744) of the *AVI* scenario is distributed as 0, 0.0004, 0 and 0.7740 - corresponding to minor, marginal, critical and catastrophic - we know that most of the *AVI* scenario risk factor(99.94%) is catastrophic, which is more severe than the risk factor 0.9745 distributed as 0, 0.5002, 0 and 0.4743, which has catastrophic risk(only 48.67%).

We have developed and solve scenario risk models for all scenarios of the pacemaker example (programming, *AVI, AAI, VVI, AAT*, and *VVT*). Table 6.3 shows how the risk factor of each scenario is distributed among the severity classes, as well as the overall scenario risk factors. Figure 6.9 presents graphically the information given. The bar's shade represents the severity class and the z-axis represents the value of the risk factor for a given severity class.

| | | Scenario Name | | | | | |
|---|---|---|---|---|---|---|---|
| | | Programming | *AVI* | *AAI* | *VVI* | *AAT* | *VVT* |
| Severity Level | Minor | 0.3196 | 0 | 0 | 0 | 0 | 0 |
| | Marginal | 0.1782 | 0.0004 | 0.5002 | 0.5002 | 0.5001 | 0.5001 |
| | Critical | 0 | 0 | 0 | 0 | 0 | 0 |
| | Catastrophic | 0 | 0.7740 | 0.4743 | 0.4743 | 0.4747 | 0.4747 |
| | Scenario risk factors | 0.4951 | 0.7744 | 0.9745 | 0.9745 | 0.9748 | 0.9748 |

Table 6.3. Distribution of the scenarios risk factors among severity classes.

Table 6.3 shows the risk distribution of the scenarios among the four severity classes. The columns in the table represent the scenarios and the rows represent the severity classes. A value in the cell $risk_{m,n}$ where *n* is the row and *m* is the column gives us the risk factor of the scenario *m* with the severity *n*. Similarly Figure 6.9 shows a 3-D bar graph of the scenario risk factors versus the four severity classes. The graph is shaded according to the distribution of the scenario risk factors among the severity classes (as shown in the legend). As we can see, the *AVI* scenario has a high catastrophic risk value. The other scenarios, like the *AAI, VVI, AAT* and *VVT,* have marginal and catastrophic risk values.

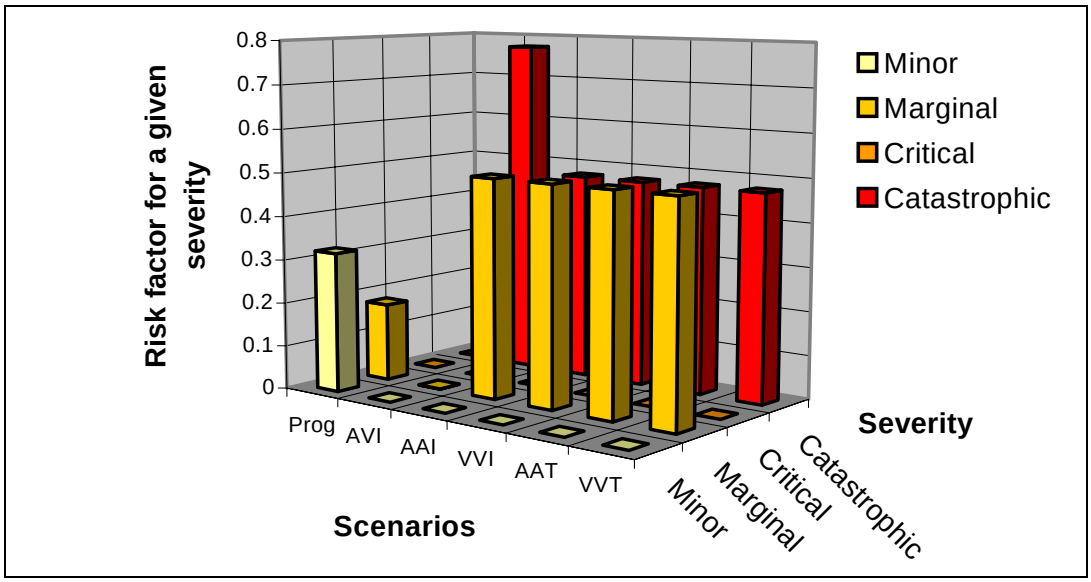



Figure 6.9 Distribution of the scenarios risk factors among severity classes.

Several observations are made from Figure 6.9. First, all scenarios from the operational mode have higher risk factors than the programming scenario, which is used just to set the mode of the pacemaker. Next, it is obvious that knowledge of the distribution of scenarios risk factors among severity classes provides valuable information for the risk analysts in addition to the overall scenario risk factor. Thus, the *AVI* scenario has the smallest scenario risk factor (0.7744) among the operational scenarios (*AVI, AAI, VVI, AAT*, and *VVT*). However, most of the *AVI* scenario risk factor belongs to the catastrophic severity class (0.7740), that is, the *AVI* scenario has the highest value of the risk factor in the catastrophic severity class. The risk factors of the other operational scenarios are distributed almost equally among the marginal and catastrophic severity classes with the values in catastrophic class significantly smaller than for the *AVI* scenario. Programming scenario has the smallest overall scenario risk

factor (0.4951) distributed only among minor and marginal severity classes, which means that it is the less critical scenario in the pacemaker case study.

### 6.6.6 Use case and overall system risk factor

Since in the pacemaker example we considered one scenario per use case, the use case risk factors are identical to the scenarios risk factors. For the pacemaker example, according to [Douglass, 1998] the inhibit modes are more frequently used than the triggered mode. Also, the programming mode is executed significantly less frequently than the regular usage of the pacemaker in any of its operational modes. Hence, we assume the probabilities for programming use case and five operational use cases (*AVI, AAI, AAT, VVI*, and *VVT*) (see chapter 7).

Using equations 6.4 and 6.5 and the use case probabilities shown in chapter 7, we estimate the overall risk factor of the pacemaker *0.9118.* The distribution of the overall system risk factor among severity classes is presented in Table 6.4 and Figure 6.10. The Table shows the distribution of the system risk factor among the four severity classes and Figure 6.10 shows the 3d bar cardiac pacemaker system risk distribution among the severity classes. As we can see most of the cardiac pacemaker system's risk falls in to the catastrophic severity class. We see that the system risk factor is mostly distributed among marginal and catastrophic severity class. Moreover, the catastrophic severity class is the dominant class for this system.

| | Minor | Marginal | Critical | Catastrophic |
|---|---|---|---|---|
| Overall system risk factor | 0.0032 | 0.3520 | 0 | 0.5566 |

Table 6.4 Distribution of the overall risk factor over severity classes

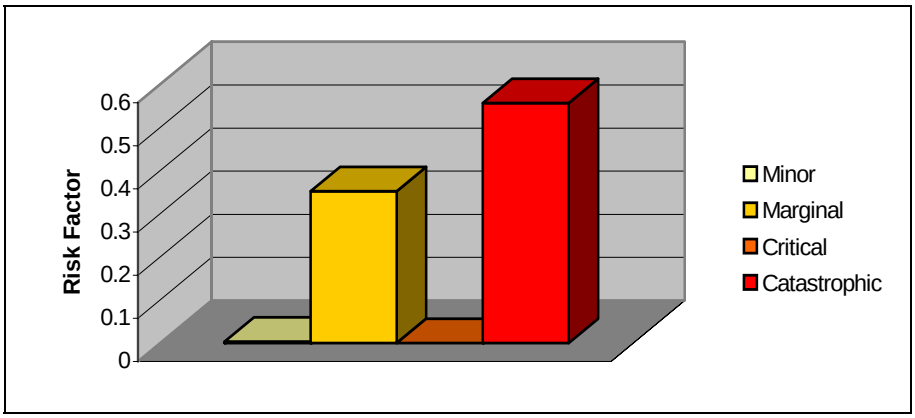


Figure 6.10 the system risk 3d bar distribution of the cardiac pacemaker.

**6.7 Sensitivity Analysis**

In the proposed methodology, we use an analytical approach and derive close form solutions. One of the advantages of this approach is that sensitivity analysis can be performed simply by plugging in different risk factor values for the components/connectors in the close form solutions, which is faster and more effective than reapplying the algorithmic solution for each set of different parameters as in [Yacoub, 2002]. Next, we illustrate the sensitivity of the scenarios and overall system risk factors to components/connectors risk factors.

Figure 6.11 illustrates the variation of the risk factor of the *AVI* scenario as a function of changes in risk factors of components active in that scenario. The variation of the risk factor of *VT* component introduces the biggest variation of the *AVI* scenario risk factor (from 0.65 to 1). This is the case because the *VT* component is the most active component in this scenario to sense the heart pulse. On the other side, the variations of the risk factor of the *AR* and *CG* components have less of an effect on the range of the variation the *AVI* scenario risk factor. However, the *AR* component is also critical since it results in the smaller value of the scenario's risk factor. Figure 6.13 shows the sensitivity of the risk factor of the programming scenario to the risk factors of the components active

in that scenario. In this case, the variation of the risk factor of the *CG* component introduces the biggest variation of the programming scenario risk factor (from 0.175 to 0.979). The variation of the overall system risk factor as a function of components risk factors is presented in Figure 6.14. It is clear that the risk factors of components *CG, VT*, and *AR* are more likely to affect the overall system risk. This is due to the fact that these components are active in scenarios that have high execution probabilities. Furthermore, the variation of the risk factors of components that are active only in the programming scenario (i.e. *RS* and *CD*) has almost no influence on the variation of the overall system risk factor because the execution probability of the programming scenario is lower than the execution probabilities of other scenarios. Figure 6.11 and Figure 6.12 show the variation of the *AVI* scenario risk factors and as a function of components and connectors risk factors. It is obvious that both *AVI* scenario risk factor and the overall system risk factor are the most sensitive to the risk factor of the *CG-VT* connector.

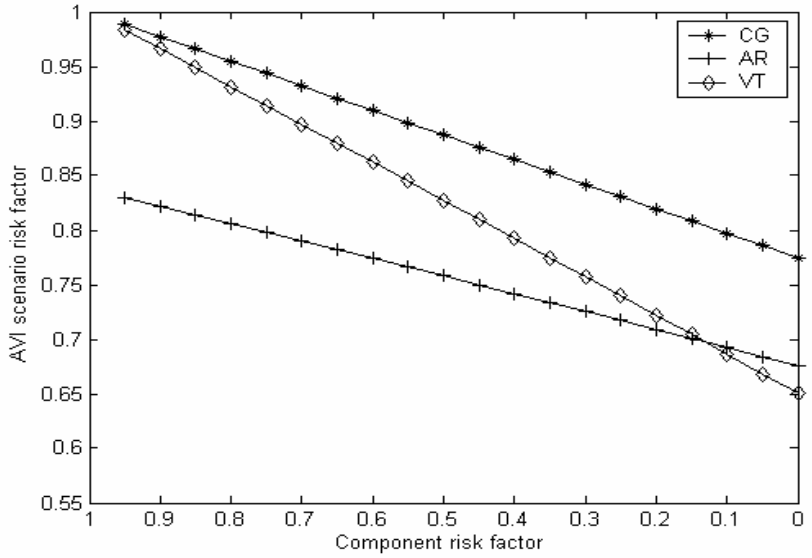



Figure 6.11 Sensitivity of the AVI scenario risk factor to the risk factor of the components

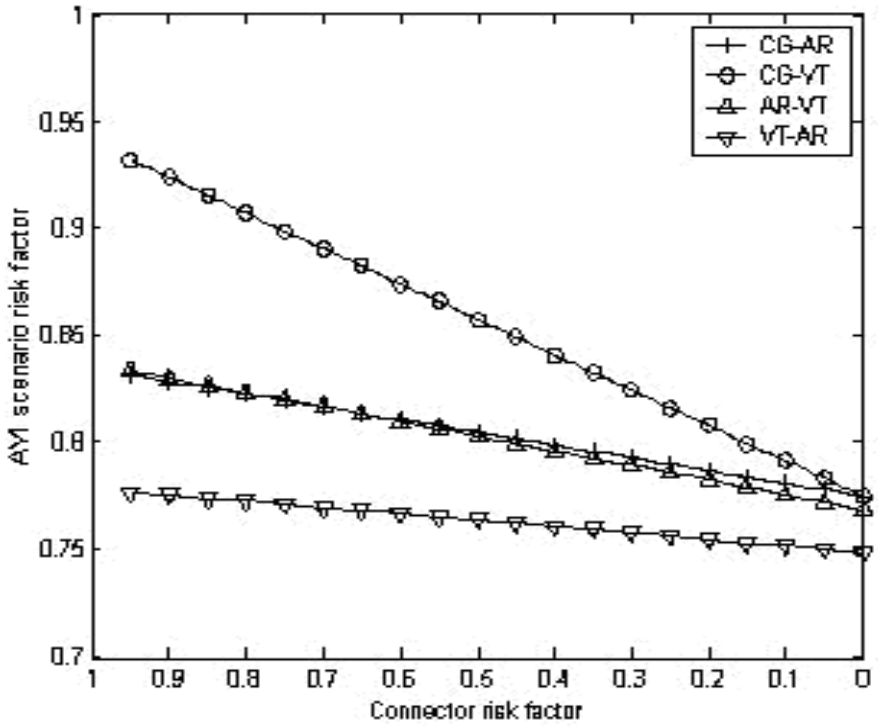



Figure 6.12 Sensitivity of the *AVI* risk factor to the risk factors of the connectors

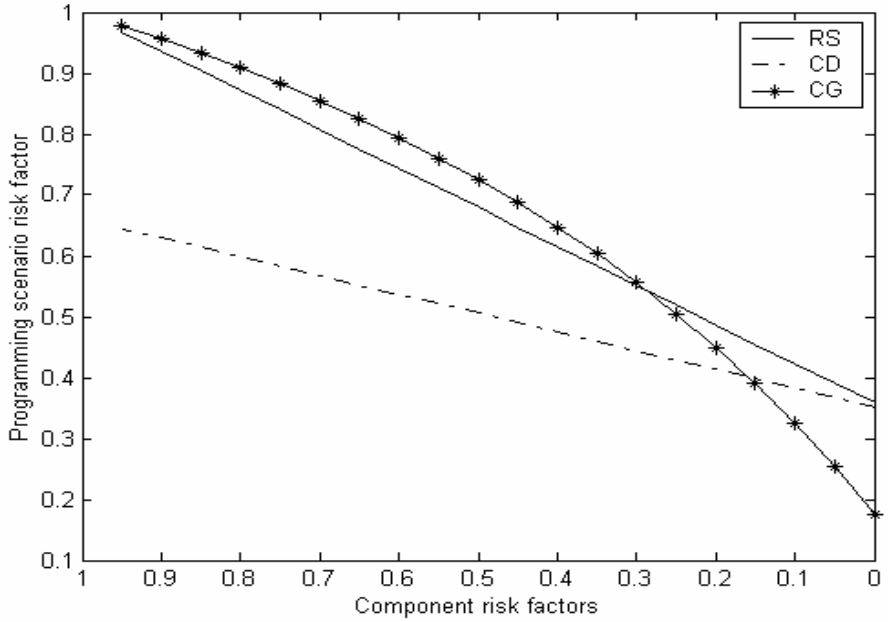



Figure 6.13 Sensitivity of the programming scenario risk factor to the risk factors of the components.

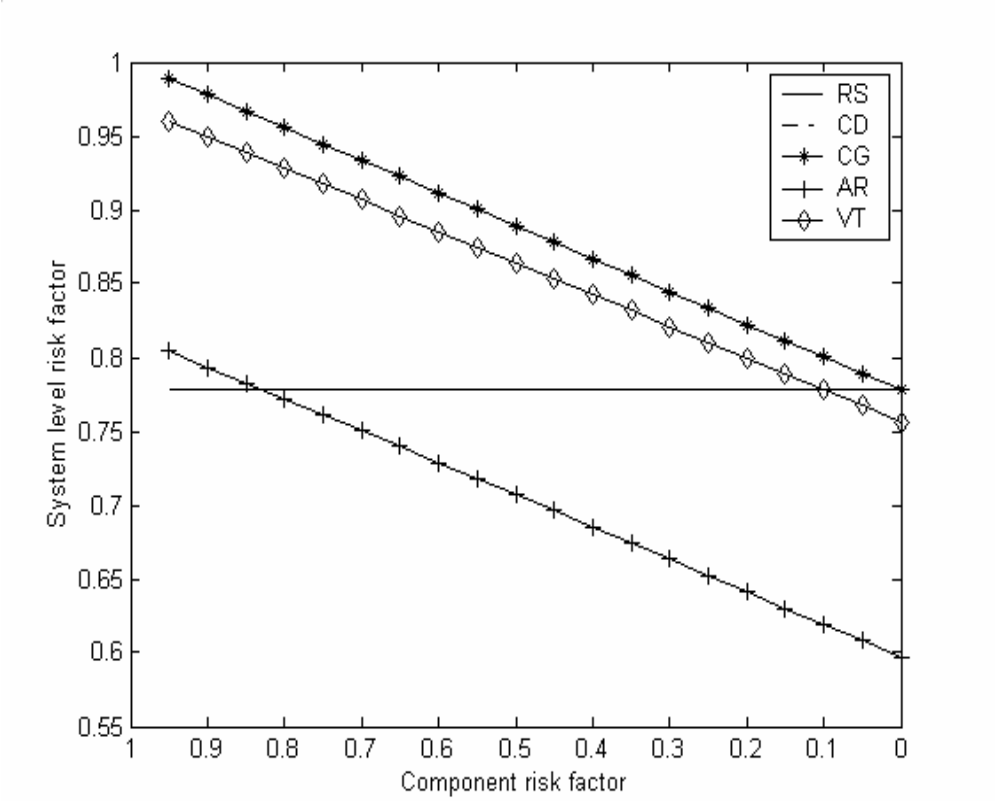



Figure 6.14 Sensitivity of the overall system risk factor to the risk factors of the components

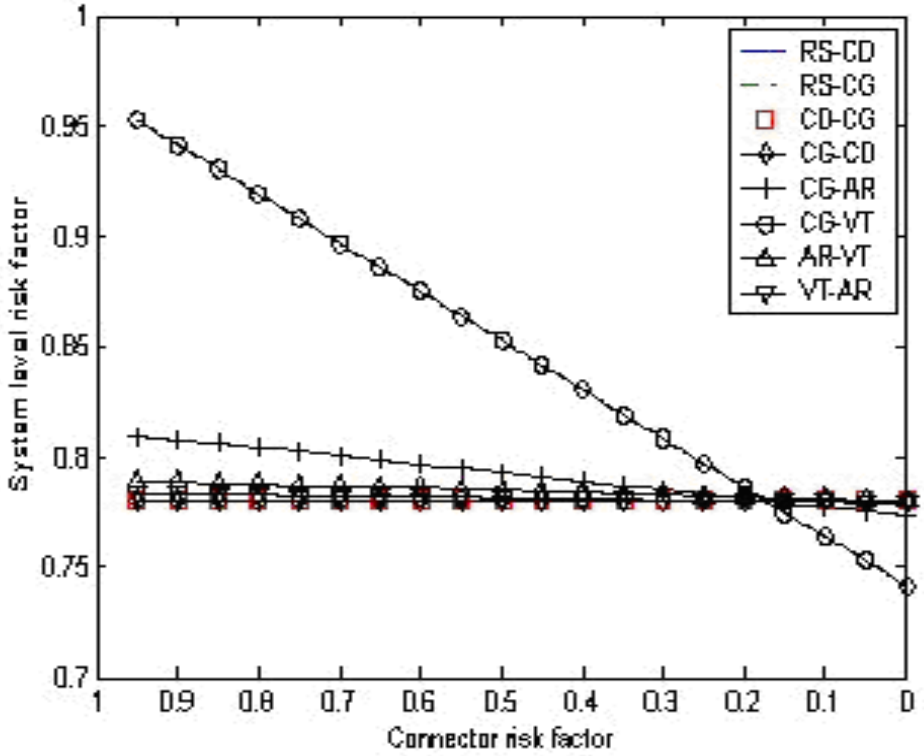



Figure 6.15 Sensitivity of the overall risk factor to the risk factors of the connectors

## 6.8 Conclusion and Future Work

This chapter presents the architectural-level reliability-based risk assessment methodology [Katerina, 2003]. In this chapter, we propose a methodology for risk assessment based on the UML specifications such as use cases and sequence diagrams that can be used in the early phases the software life cycle. Our methodology uses dynamic complexity and dynamic coupling metrics obtained from the UML specifications [Hassan, 2001]. The risk assessment methodology presented in this chapter considers both component and connector risk factors. It is used for calculating the risk factors of various components and connectors and estimating a risk factor of scenarios, use case and system level. We combine severity [Hassan1, 2003], [Hassan2, 2003], [Hassan, 2005] and complexity (and coupling) metrics to obtain risk factors for the components (and connectors). It aggregates the risk factors of components/connectors to calculate risk factors of scenarios among the various severity classes by solving the Markov chain. Using the scenarios probabilities, the use case risk factors could be estimated. The system-level risk factor is calculated by averaging the use case risk factors with their execution probabilities. The risk methodology is applied to the cardiac pacemaker case study; we have also applied it to the HCS case study (see chapter 7). Since the methodology is entirely analytical and provides a closed form solution, it is very suitable for sensitivity analysis and automation. In fact, a prototype of the risk assessment tool written in JAVA which reads the embedded UML information from Rational Rose, and calculates the various risk factors has already been developed (see chapter 9) [Wang, 2003].

In summary our methodology consists of the following:

(1) Accurate and more efficient methods to estimate risk factors on different levels and

(2) Additional information useful for risk analysis.

(3) Estimation of overall system risk factor

(4) Estimation of scenarios and use cases risk factors which enable us to focus on the high-risk scenarios and uses cases even though they may be rarely used and therefore not contributing significantly to the overall system risk factor.

(5) Estimation of the distribution of the scenarios/use cases/system risk factors over different severity classes which allow us to make a list of critical scenarios in each use case, as well as a list of critical use cases in the system.

(6) Finally, we identify a list of critical components and connectors that has high risk values in high severity classes.

Our future work is focused on generalization of the methodology presented in this chapter. We will consider different kinds of dependencies that might be present in the UML use case diagrams (i.e considers the various relationships between the use cases) and the way to derive their risk factors. Another direction of our future research is the development of performance, maintainability based risk assessment methodology.

# *Chapter 7*

## Case Studies

### 7.1 Command and Control System (CCS)

The case study we use to applay our methodology is a large command and control system that is used in a life-critical, mission-critical application. This system was modeled using the Rational Rose Realtime CASE tool [Rose, 2001]. The CCS is a Computer Software Configuration Item (CSCI) that provides the following functions: This CCS system is a complex system with operations setting controller, fault recovery procedures, and pump control functionalities. The CCS is responsible for providing overall management of pumps as well as performing the necessary monitoring and response to sensors data. Also, it is responsible for performing automated startup, and controlling Thermal System reconfigurations. During each execution cycle, a check is performed for incoming commands. Received commands are validated in the same execution cycle. Mode change commands, which will reconfigure the Internal Thermal System, are also accepted from other components of CCS to compensate for system component failures or coolant leaks. A failure recovery system detects failure conditions and performs recovery operations in response to the detected failures. Failure conditions include combinations of Pump failures and Shutoff Valve failures. The system has a hierarchical architecture. The top-level software architecture of this system is shown in use case diagram Figure 7.1.

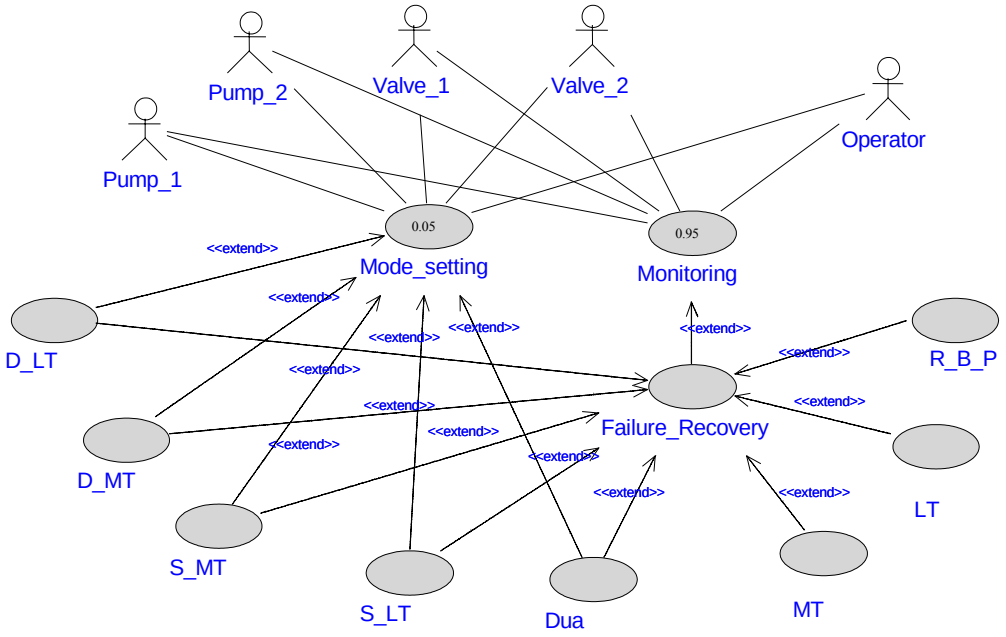



Figure 7.1 use case diagram of the CCS

Every use case is mapped to one scenario. Table 7.1 shows the probability of execution of each scenario.

| Use case (Scenario) | Probability |
|---|---|
| S_LT | 0.01 |
| D_LT | 0.01 |
| MT | 0.01 |
| S_MT | 0.01 |
| D_MT | 0.01 |
| R_B_P | 0.01 |
| Dua | 0.01 |
| LT | 0.01 |
| Monitor | 0.92 |

Table 7-1 The probabilities of the scenarios of the CCS

Figure 7.2 shows the *Dua* sequence diagram. This sequence diagram is an implementation of the *Dua* use case.

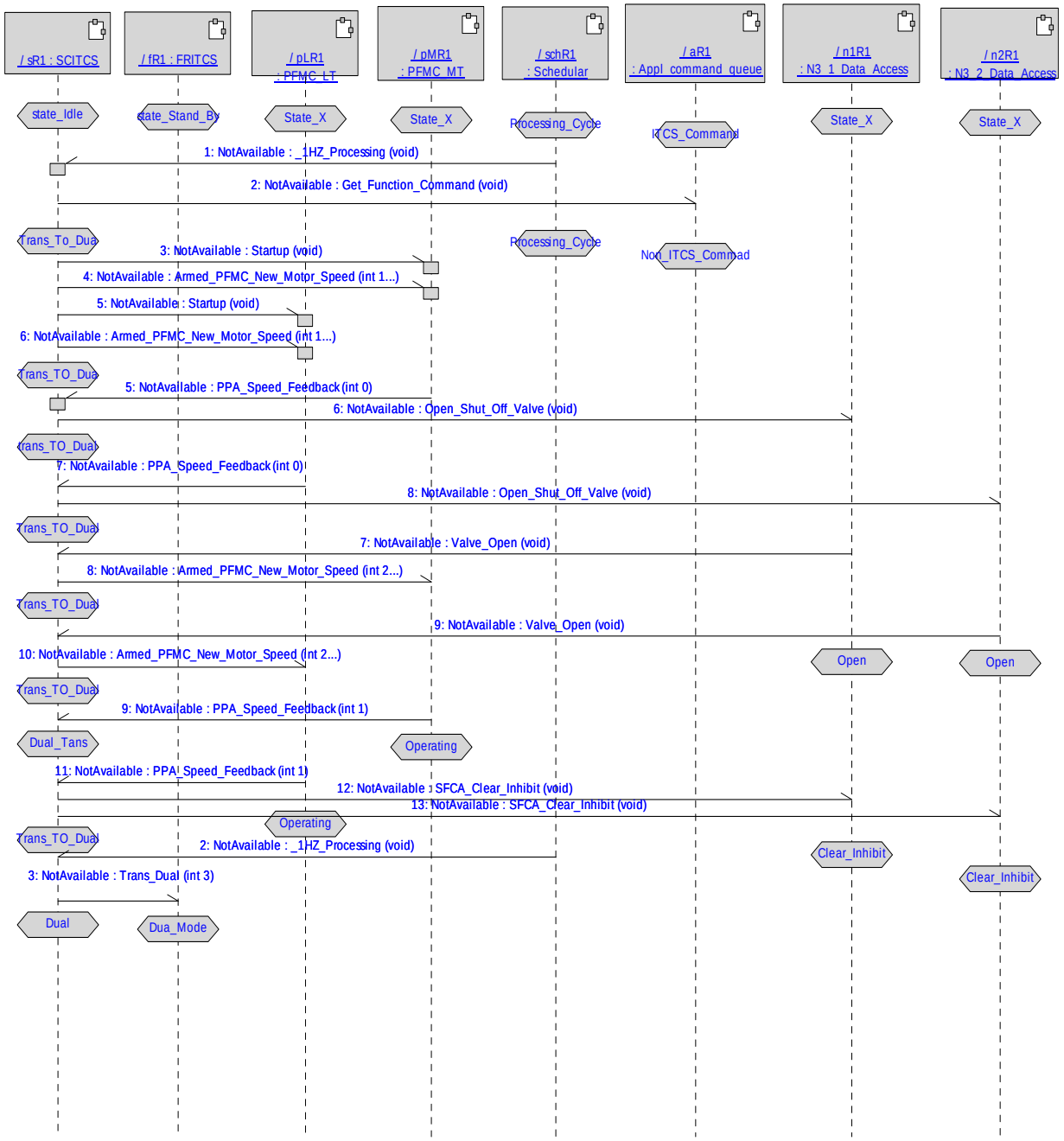



Figure 7.2 *Dua* sequence diagram

Table 7-2 shows the risk factor and severity of each component in the *Dau* scenario. From this table we can identify that component *pFMC_LTR1 is the most sever component*

| scenario | component | complexity | severity | Risk Factor |
|---|---|---|---|---|
| *Dua* | fRITCSR1 | 0.0625 | 0.75 | 0.046875 |
| | pFMC_MTR1 | 0.125 | 0.75 | 0.09375 |
| | appl_command_queueR1 | 0.0625 | 0.95 | 0.059375 |
| | sCITCSR1 | 0.375 | 0.95 | 0.35625 |
| | n3_2_Data_AccessR1 | 0.0625 | 0.5 | 0.03125 |
| | n3_1_Data_AccessR1 | 0.0625 | 0.75 | 0.046875 |
| | pFMC_LTR1 | 0.125 | 0.75 | 0.09375 |
| | schedularR1 | 0.125 | 0.25 | 0.03125 |

Table 7-2 *Dua* Scenario components risk factor

The other sequences diagrams of this case study and the detailed results are shown in Appendix B.

### 7.1.1 CCS Scenarios risk factors

Table 7-3 shows the distribution of risk factor of the scenarios among severity classes.

| | Minor | Marginal | Critical | Catastrophic | Risk Factor |
|---|---|---|---|---|---|
| S_LT | 0 | 0.001423 | 0.224649 | 0.084604 | 0.310676 |
| D_LT | 0.095979 | 0 | 0.06582 | 0.31299 | 0.474789 |
| MT | 0 | 0.020362 | 0.208994 | 0.386209 | 0.615565 |
| S_MT | 0 | 0.00218 | 0.143619 | 0.501907 | 0.647706 |
| D_MT | 0.025382 | 0 | 0.413158 | 0.000468 | 0.439008 |
| R_B_P | 0.19015 | 0.043739 | 0 | 0.039977 | 0.273866 |
| Dua | 0 | 0.140222 | 0.000542 | 0.35643 | 0.497194 |
| LT | 0.286245 | 0.055764 | 0 | 0 | 0.342009 |
| Single_LT | 0 | 0.001423 | 0.224649 | 0.084604 | 0.310676 |
| Monitor | | | 0.341113 | 0.201009 | 0.542122 |

Table 7-3 The risk factor for every scenario

Table 7-3 shows that scenario S_MT has total risk factor 0.647706 and it has 0.501907 catastrophic risk factor. It looks that this is the most critical scenario in the system.

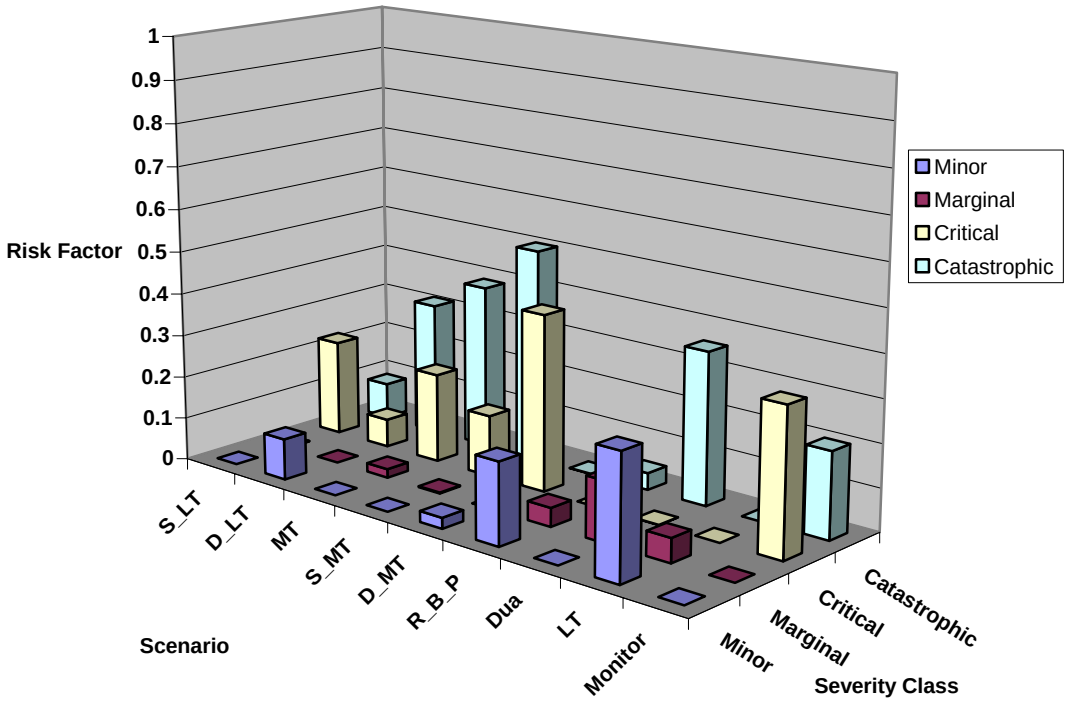



Figure 7.3 the 3D-bar graph for the distribution of scenarios risk factor among severity classes.

### 7.1.2 CCS System Risk Factor

Table 7-4 shows the distribution of the system risk factor between the four severity classes. It shows also from Figure 7.4 that the risk of failure of the system is critical. The system has 0.537867 total risk factor and most of this risk factor is critical which is 0.326638.

| | Minor | Marginal | Critical | Catastrophic | Risk Factor |
|---|---|---|---|---|---|
| System | 0.005978 | 0.002651 | 0.326638 | 0.2026 | 0.537867 |

Table 7.4 system risk factor and the distribution of the risk factor

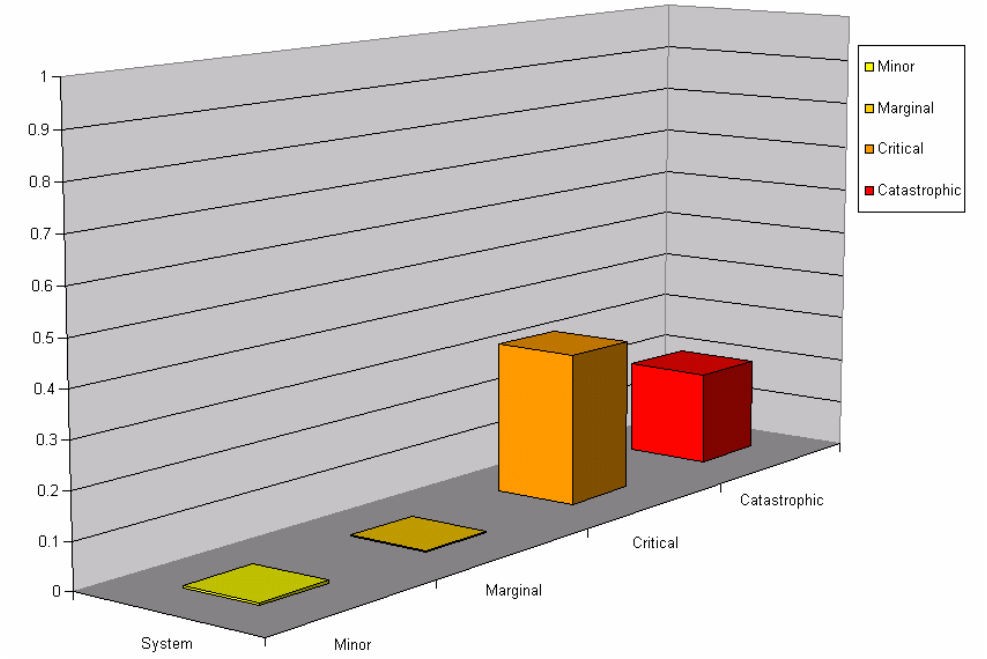


Figure 7.4 3D-bar of risk factor distribution for the HCS system

In this section and also in the section 7.3 we show the results of applying the severity analysis methodology on other case studies.

## 7.2 Remote Transmission System (RTS)

The RTS involves a controller and a remote system. The controller sends commands and receives data from the remote system. Transmission takes place in two different modes. Figure 7.5 shows the Use case view of the system. The actor "*Operator*" represents the operator of the controller. The Remote system is shown as an actor as well. The two different transmission modes are represented as use cases "*TransmitA*" and "*TransmitB*". The "*Handle Transmission Failure*" is used whenever a transmission fails.

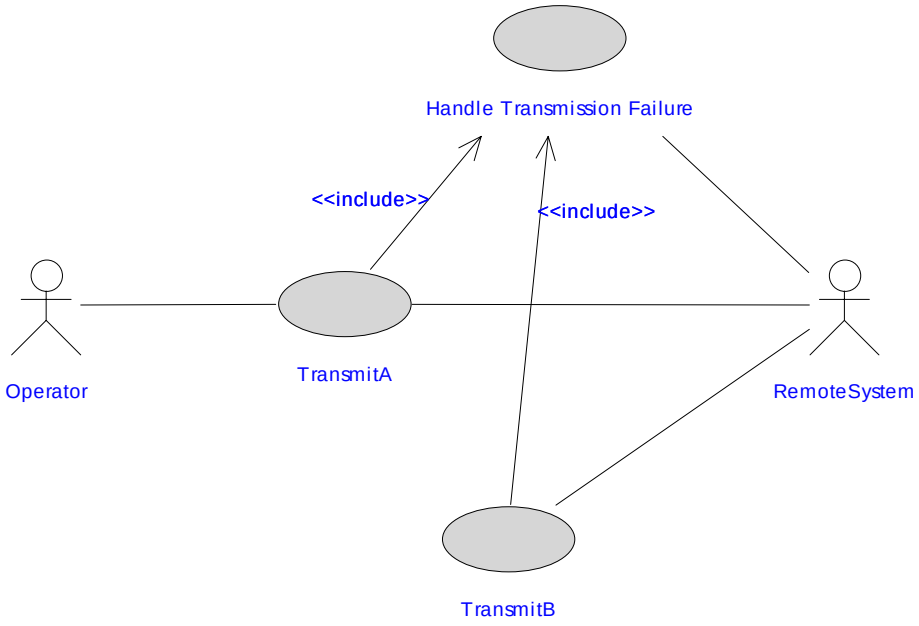



Figure 7.5 Use case diagram for RTS case study

Figure 7.6 gives the system level sequence diagram for one of the scenarios of "*TransmitB*" use case. *Ac1* is an external actor that stores the current status of the remote system. Based on the status, a command is sent to the remote system. *S1*, *S2* and *S3* represent the system states.

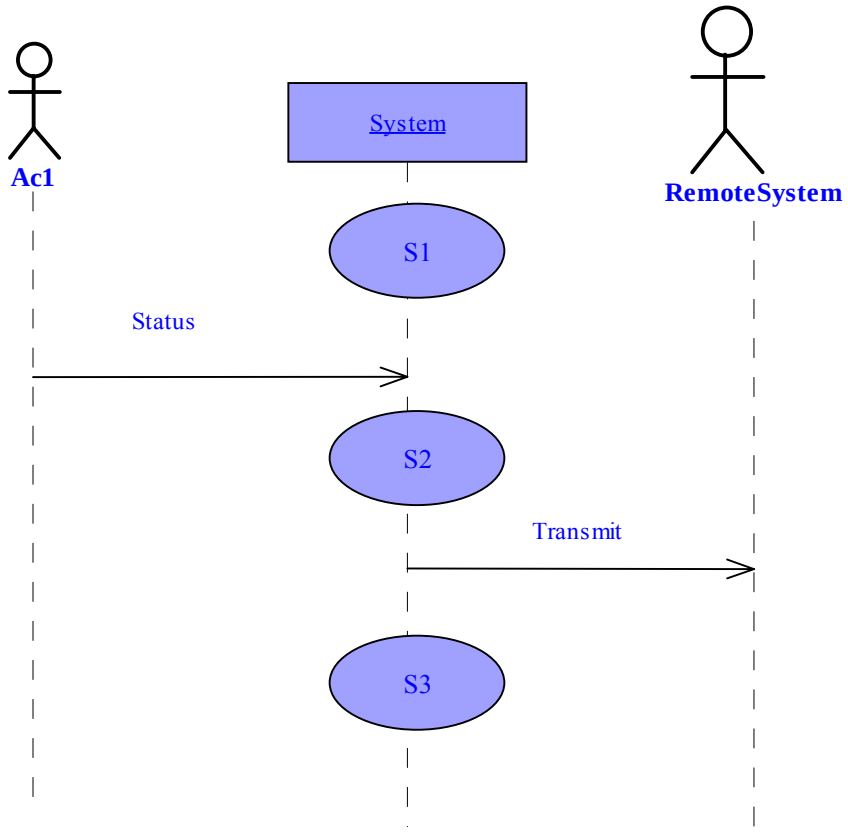



Figure 7.6 System level sequence diagram of *TransmitB* use case

The following table shows the list of events (signals/messages) sent between the System and the external actors. The FFA guidewords are applied to these external events. Since we only consider the performance related failures [Cortellessa, 2005], we apply only the "*LATE*" guideword. The FFA table obtained is shown in Table 7-6.

| **Event Name** | **Actor sends or receives this event** |
|---|---|
| *Status* | From A1 to system |
| *Transmit* | From system to Remote System |

Table 7-5 External events

| Event | Guide-word | Failure | Effect | Cost | Severity |
|---|---|---|---|---|---|
| *Status* | *Late* | A1 takes longer time to send the status | The input cannot be handled at the right time | Mission Loss | 0.95 |
| *Transmit* | *Late* | Commands don't reach the remote system at proper time | Commands cannot be transmitted in time | Mission Loss | 0.95 |

Table 7-6 FFA Table for *TransmitB* scenario

This results show the severity analysis for the system scenario level.

### 7.3 E-Commerce system

The e-commerce application allows customers and suppliers to interact with each other over the Internet. In this type of applications long response times may easily lead customer to change the supplier, with consequent damages such as loss of money and market. The severity of performance failures (i.e., violations of performance requirements) depends on the type of failure and usually is different for different types of failures.

The e-commerce system allows a customer to browse through the various catalogs provided by the suppliers, select the item to be purchased, and place the order. The order is validated by checking that the customer has a contract with the supplier and one or more bank accounts through which payments can be made. The supplier checks for the availability of the product, and if available, ships the product. After received the product, the customer sends back an acknowledgement. Finally, the invoice is processed by electronically transferring funds from the customer's bank account to the supplier's bank account. The Use case model of the e-commerce application is shown in Figure 7.7.

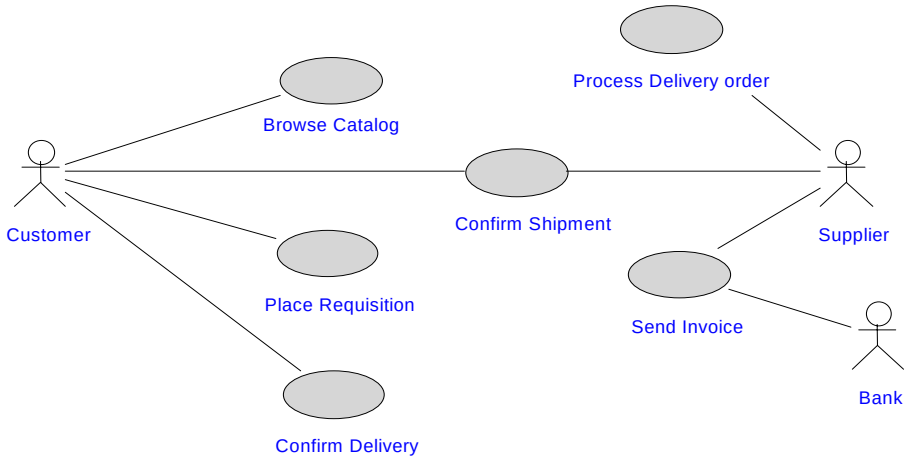



Figure 7.7 Use case view of e-commerce system

For illustration purposes, we consider the *Place Requisition* scenario, where the customer places an order and receives a receipt of the order. The Sequence diagram is shown in Figure 7.8.

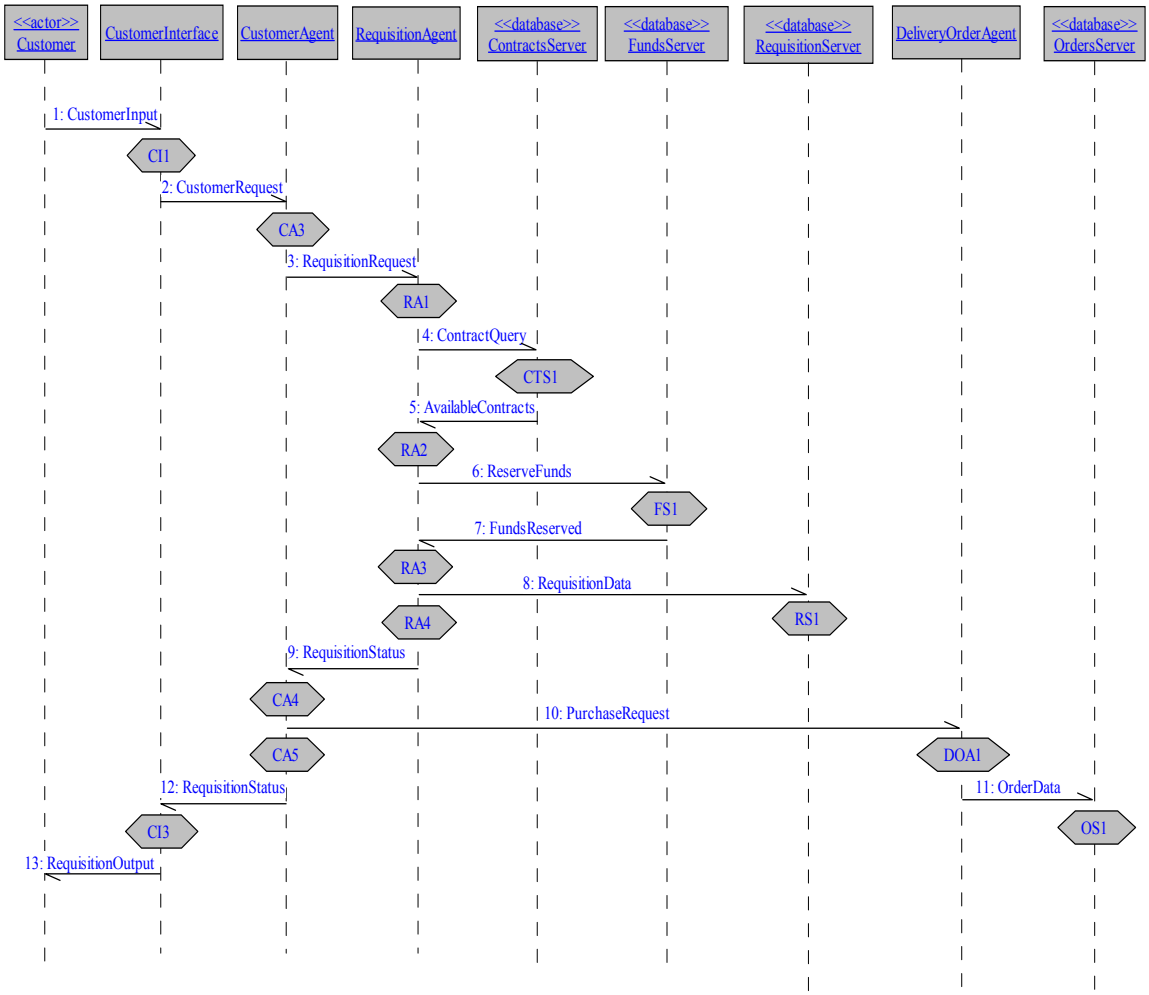



Figure 7.8 Sequence Diagram for the Place Requisition scenario

The system level sequence diagram is shown in Figure 7.8. It shows the messages that are exchanged between the system and the external actors involved. The FFA guidewords are applied to these external events Figure 7.9. Since we consider only the performance risk for this case study, we apply only the "*LATE*" guideword.

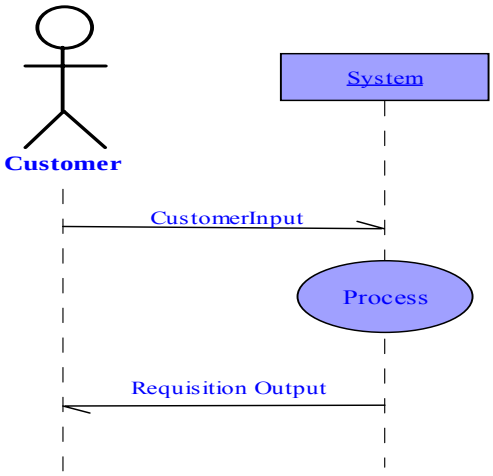



Figure 7.9 System level sequence diagram for Place Requisition scenario

After we apply the FFA we come up with the FFA Table 7-7, which show the severity of failure of the scenario [Cortellessa, 2005] Figure 7.9.

| Scenario | Event | Guide words | Failure | Effect | Cost | Severity |
|---|---|---|---|---|---|---|
| Place Requisition | Customer input | *Late* | n/a | | | |
| | Request output | *Late* | The confirmation message for the order placed takes a long time to be displayed to the customer | • Customer's time is wasted.<br>• The customer gets impatient.<br>• The customer might cancel the order.<br>• The customer might not order again. | Mission loss | 0.95 |

Table 7-7 FFA Table for *Place Requisition* scenario

This table shows the results of applying the severity methodology for system scenario level and the table shows the FFA output.

# *Chapter 8*

## Validation Criterion

### 8.1 Introduction

In this chapter we explore two ways to validate the proposed risk methodology and the proposed severity assessment methodology. First we use validation criteria similar to criteria explored in [Elemam, 2000] (section 8.2). Second we compare the results of risk factors predicted using the proposed methodology and risk factors resulted from the simulation model [Yacoub, 2002] and we looking for correlation (Section 8.3). And in section 8.4 we validate the severity analysis methodology using the validation criteria proposed in section 8.2.

We proceed in this chapter as follows: Section 8.1 introduction, Section 8.2 is the proposed validation criteria, Section 8.3 is validation by comparison, and Section 8.4 is the proposed severity methodology validation. We conclude with Section 8.5.

### 8.2 Validation criterion

To validate the proposed risk model, we use criteria that are based on the analysis of categorical data [Kohavi, 1998], [Briand, 2000], [Lanubile, 1997]. In our validation, we assume that we have two variables, real risk and predicted risk, with only two discrete values, low and high risk. We restrict ourselves to binary risk classes: High and Low. Thus the data can be represented by a two-dimensional table, shown in Table 8.1, with one row for each level of the variable real risk and one column for each level of the variable predicted risk. In the following section we will explain the notations used.

### 8.2 1 Confusion matrix

A commonly used notation for presenting risk data for evaluation is the Confusion matrix, Table 8.1 [Kohavi, 1998], [Briand, 2000]. This matrix contains information about actual risk done by the simulation model and predicted risk done by the proposed model. Such a confusion matrix also appears frequently in the medical sciences in the context of evaluating diagnostic tests; for example, see [Gordis, 1996].

The data can be represented by a two-dimensional table [Elemam, 2000] Table 8-1. With rows for real risk (low, high) and columns for predicted risk (low, high). In our context, the first row contains low risk components, while the second row contains high risk components. The first column contains components that the models classify as low risk, while the second column contains components classified as high risk.

| | | *Predicted Risky Components* | | |
|---|---|---|---|---|
| | | *Low* | *High* | |
| *Real Risky Components* | *Low* | $n_{11}$ | $n_{12}$ | $N_{1+}$ |
| | *High* | $n_{21}$ | $n_{22}$ | $N_{2+}$ |
| | | $N_{+1}$ | $N_{+2}$ | $N$ |

Table 8-1 Confusion matrix.

### 8.2.2 Definition of the terms used in the confusion matrix

$n_{11}$ The number of components which have low risk factor; it is predicted as having low risk factor.

$n_{12}$ The number of components which have low risk factor; it is predicted as having high risk factor.

$n_{21}$ The number of components which have high risk factor; it is predicted as having low risk factor.

$n_{22}$ The number of components which have high risk factor; it is predicted as having high risk factor.

$N_{1+}$ Number of low risk factor components.

$N_{2+}$ Number of high risk factor components.

$N_{+1}$ The number of components predicted as Low risk

$N_{+2}$ The number of components predicted as High risk

The number of all components is

$$N = N_{1+} + N_{2+} = N_{+1} + N_{+2} \qquad 8.1$$

Having data represented by a confusion matrix, we could measure the validity of the proposed model. To study the predictive validity of the proposed model we use the evaluation criteria explained in the following section.

A number of different measures are used to evaluate binary classifiers in software engineering research [Almeida, 1998], [Lanubile, 1997]. An intuitively obvious way to evaluate the overall "goodness" of a classifier is to calculate the proportion of correct classifications. The proportion correct has been used in a number of previous studies to evaluate classifiers [Elemam, 2000], [Almeida, 1998], [Lanubile, 1997], [Schneidewind, 1994]. The misclassification rate quality achieved [Lanubile, 1997] is also used in engineering research as a criteria for evaluating predictive models. Different authors use different measures; they also often report multiple measures in a single study [Elemam,

2001b]. We recommend the use of multiple measures for a single study as it may be useful for getting an overall intuitive picture of prediction performance.

In the following section, we will explain the proposed criteria that could be used to evaluate the proposed risk model.

### 8.2.3 Proportion correct

Most authors proposed the proportion correct value, for example [Almeida, 1998], [Lanubile, 1997], [Schneidewind, 1994]. Porter in [Porter, 1993], called the proportion correct value as "correctness value". This is an intuitively interesting measure of prediction performance since it is easy to interpret and is defined as:

$$A = \frac{n_{11} + n_{22}}{N} \qquad 8.2$$

It gives the ratio of components that the model correctly predicts.

### 8.2.4 Misclassification rate

This evaluative measure is used in a number of different studies, such as [Khoshgoftaar, 1995a], [ [Koshgoftaar, 1996b], [Elemam, 2001b], [Khoshgoftaar, 1997]. Two types of errors could take place using binary classifiers. A Type 1 error is made when a high risk component is classified as low risk, while a Type 2 is made when a low risk component is classified as high risk. It is desirable to have both types of error be small. However, since the two types of errors are not independent, software engineering managers should consider their different implications. As a result of a Type 1 error, an actual high risk component could pass quality control. This would cause the release of a lower quality product and more fix effort when a failure takes place. As a result of a Type 2 error, an actual low risk component will receive more testing and inspection effort than

needed. Type 1 and Type 2 errors are a function of $n_{21}$ and $n_{12}$. We use the following measures of misclassification [Schneidewind, 1994], [El-Emam, 1999c].

The Type 1 misclassification rate is $1-f$, and the Type 2 misclassification rate is $1-S$ [Khoshgoftaar, 1999], [Taghi, 1997]. Where $f$ is the specificity of the model, it is the proportion of low risk components that have been correctly classified as low risk components, [Almeida, 1998], [El-Emam, 1999c], and it is defined as follows:

$$f = \frac{n_{11}}{n_{11} + n_{12}} \qquad 8.3$$

Also, $S$ is the sensitivity of the model, and is the proportion of high risk components that have been correctly classified as high risk components [Briand, 1993a], [Porter, 1990], and is defined as follows:

$$S = \frac{n_{22}}{n_{21} + n_{22}} \qquad 8.4$$

Some authors, such as [El-Emam, 2000] report the sensitivity and specificity values directly as measures. Ideally, both the sensitivity and specificity should be high. A low specificity means that there are many low risk components that are classified as high risk. Therefore, the organization would be wasting resources reinspecting or focusing additional testing effort on these components. A low sensitivity means that there are many high risk components that are classified as low risk. Therefore, the organization would be passing high risk components to subsequent phases. In both cases the consequences may be expensive field failures or costly defect correction later in the life cycle.

### 8.2.5 Quality achieved

We are interested in measuring how effective the model is in terms of the quality achieved after the components classified as high risk have undergone an extra verification activity. If all the high risk components are properly classified, all mitigation actions will take place by extra verification, and perfect quality will be achieved. However, quality will be degraded with each high risk component that is not identified. We could measure the criterion of achieved quality using the COmpleteness (CO) measure [Briand, 1993] which is the percentage of high risk components that have been actually classified as such by the model.

$$. \; CO = \frac{n_{22}}{N_{2+}} \qquad (5)$$

Based on the time frame available and also the availability of data we will use the simulation model to evaluate the proposed model. We will compare the results which the proposed model predicts with the results of the simulation of the Pace maker case study. Finally we will use the simulation model results to validate the proposed model.

## 8.3 Pacemaker case study

To evaluate our proposed risk model, the proposed evaluation criteria are applied on the results for system level component and connector risk factors.

### 8.3.1 Confusion matrix for components

Table 8-2 shows the confusion matrix representing results from system level component risk factor. The predicted risk in the columns represents the result from the proposed model. The Real risk in the rows represents the simulation model results. The value 3 corresponding to the "Low" row and "Low" column (cell $n_{11}$) shows that three components that were predicted to be low risk components by our methodology prove to

be low risk components in the simulation results as well. Similarly the value in cell $n_{22}$ corresponding to the "High" row and "High" column shows that two components that were predicted to be high risk by our proposed methodology prove to be high risk components in the simulation results.

| | | Predicted Risk Components set | | |
|---|---|---|---|---|
| | | Low | High | |
| Real Risk Components set | Low | 3 | 0 | 3 |
| | High | 0 | 2 | 2 |
| | | 3 | 2 | 5 |

Table 8-2 the confusion matrix for the results System Level Component Risk Factor

***Proportion Correct***

The following calculation shows the proportion of correct predictions obtained from our methodology when compared to the simulation results. The result is 1, which means that our methodology predicted results are 100% identical to the simulation model results for the pace maker case study. The result might not be this accurate for other case studies, which involve many more components than the pace maker example.

$$A = \frac{n_{11} + n_{22}}{N} = 5/5 = 1$$

***Misclassification rate***

The following shows the calculation of the misclassification rate. There are 0 misclassifications in Type 1 and Type 2.

$$f = \frac{n_{11}}{n_{11} + n_{12}} = 3/(3+0) = 1$$

The Type 1 misclassification rate is $1-f$ = 1-1 = 0

Type 2 misclassification rate is $1-S$ = 1 – 1 = 0, where

$$S = \frac{n_{22}}{n_{21} + n_{22}} = 2/(0+2) = 1$$

***Quality achieved***

The quality achieved is calculated as follows:

$$CO = \frac{n_{22}}{N_{2+}} = 2/2 = 1$$

### 8.3.2 Confusion matrix for connectors

The confusion matrix for system level connector risk factors is shown in Table 8-3. The cell $n_{12}$ shows that two connectors that were predicted as high risk were actually low risk connectors from the simulation results. Similarly, the cell $N_{21}$ shows that one connector that was predicted as high risk by our methodology was actually a low risk connector in the simulation results.

| | | Predicted Risk Connectors set | | |
|---|---|---|---|---|
| | | Low | High | |
| Real Risk Components set | Low | 4 | 2 | 6 |
| | High | 1 | 1 | 2 |
| | | 5 | 3 | 8 |

Table 8-3 Confusion matrix for connectors

The proportion of correct predictions and the misclassifications are calculated as in the previous section. The quality achieved is 0.5.

***Proportion Correct***

$$A = \frac{n_{11} + n_{22}}{N} = 5/8 = 0.625$$

***Misclassification rate***

$$f = \frac{n_{11}}{n_{11} + n_{12}} = 4/(4+2) = 0.667$$

The Type 1 misclassification rate

$$1 - f = 1\text{-}1 = 0.333$$

Type 2 misclassification rate is $1 - S = 1 - 0.5 = 0.5$

$$S = \frac{n_{22}}{n_{21} + n_{22}} = 1/(1+1) = 0.5$$

***Quality achieved***

$$CO = \frac{n_{22}}{N_{2+}} = 1/2 = 0.5$$

The model does not work well for the prediction of connector risk, and quality achieved, while Type 2 misclassification rate is poor.

**8.4 Validation by comparing the simulation and predicted results**

Table 8-4 shows the comparison between the predicted system level risk factors for the software components of Pace maker example, obtained from our proposed methodology and the results from the simulation model published by Yacoub et al [Yacoub, 2002]. The comparison is graphically represented in Figure 8.1. The results strongly correlate with each other and the correlation factor is 0.997.

| Component | RS | CD | CG | AR | VT |
|---|---|---|---|---|---|
| Simulation Model Result | 0.0005 | 0.00325 | 0.0025 | 0.95 | 0.91485 |
| Proposed model Result | 0.002 | 0.002375 | 0.0034 | 0.95 | 0.817 |
| | | Correlation 0.99735215 | | | |

Table 8-4 System Level Component Risk Factor and correlation

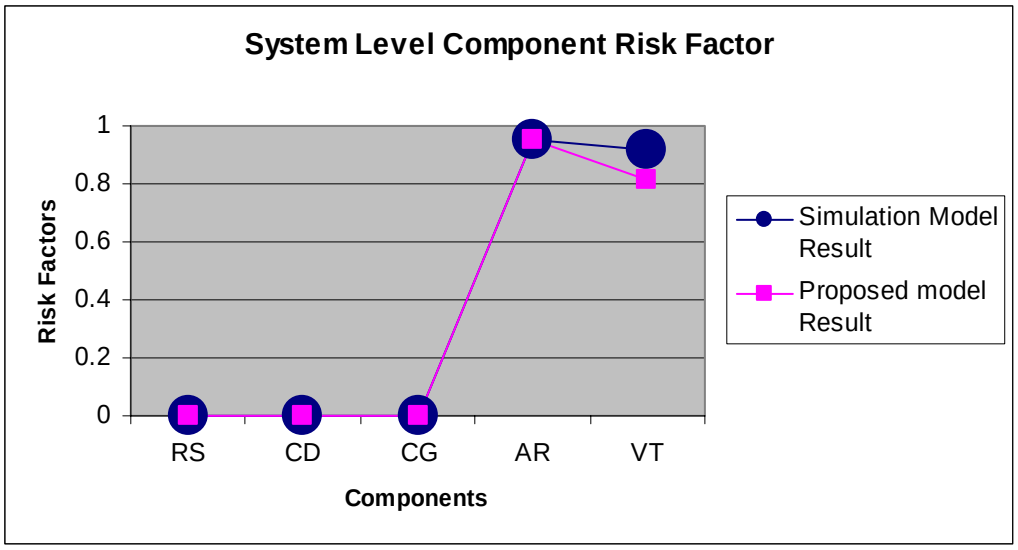



Figure 8.1 System Level Component Risk Factor

### 8.4.1 System Level Connector Risk Factor

Table 8-5 shows the comparison of predicted software connector risk factors at system level with the simulation model results by Yacoub et al [Yacoub 2002] for Pace maker case study. The comparison is graphically represented in Figure 8.2. These results show a low correlation. The correlation factor is 0.379. The simulation model considers the heart and the programmer as components, whereas in our proposed methodology, these entities are considered as external actors. Therefore, the connectors that involve the heart or the programmer are not considered in our case and hence could not be compared with the simulation results. This could be an apparent reason for the low correlation.

| Connector | RS-CD | RS-CG | CD-CG | CG-CD | CG-AR | CG-VT | AR-VT | VT-AR |
|---|---|---|---|---|---|---|---|---|
| Simulation Model Result | 0.00035 | 0.00035 | 0.00075 | 0.0005 | 0.0007 | 0.0007 | 0.2375 | 0.2565 |
| Proposed model Result | 0.000893 | 0.000893 | 0.002675 | 0.002675 | 0.5 | 0.5 | 0.0763 | 0.708 |
| | | | Correlation 0.378665 | | | | | |

Table 8-5 System Level Connector Risk Factor and correlation

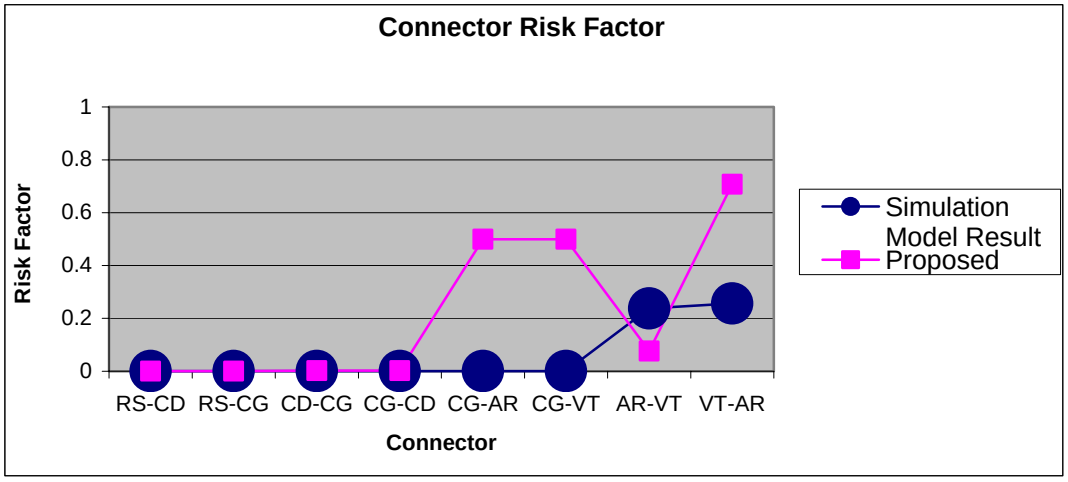



Figure 8.2 System Level Connector Risk Factor

### 8.4.2 Scenario Level component risk factor

Tables 8-6 to 8-11 show the comparison of our results with the simulation model results for scenario level component risk factor for each scenario in the Pace maker case study. Figure 8.3 to Figure 8.9 graphically represent these results. The results have a high correlation factor. The correlation values are given in the corresponding tables.

***Programming scenario***

| Component | RS | CD | CG | AR | VT |
|---|---|---|---|---|---|
| Simulation Model Result | 0.02075 | 0.1685 | 0.1215 | 0 | 0 |
| Proposed model Result | 0.05 | 0.125 | 0.15 | 0 | 0 |
| | | Correlation 0.925109675 | | | |

Table 8-6 Programming scenario Components Risk Factor

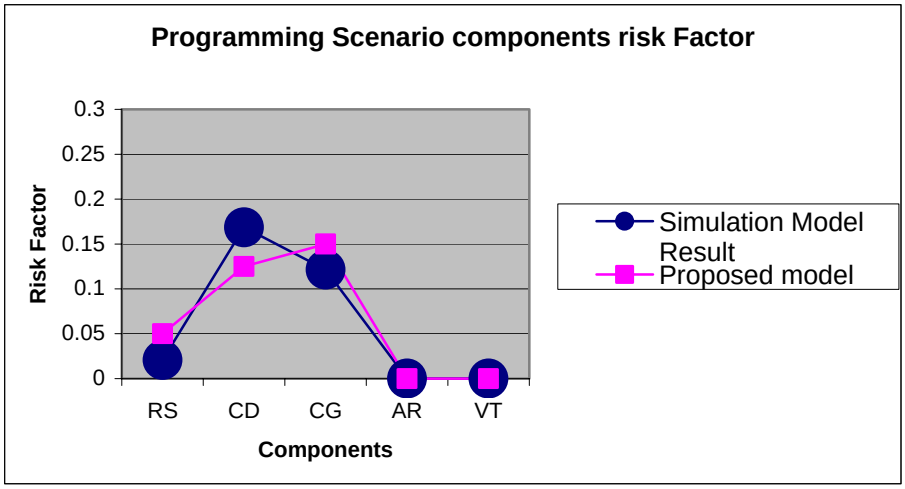



Figure 8.3 *Programming* scenario Components Risk Factor

***AVI Scenario***

| Component | RS | CD | CG | AR | VT |
|---|---|---|---|---|---|
| Simulation Model Result | 0 | 0 | 0 | 0.5054 | 0.4446 |
| Proposed model Result | 0 | 0 | 0.00016 | 0.30068 | 0.33095 |
| | | Correlation 0.98138 | | | |

Table 8-7 *AVI* scenario Components Risk Factor

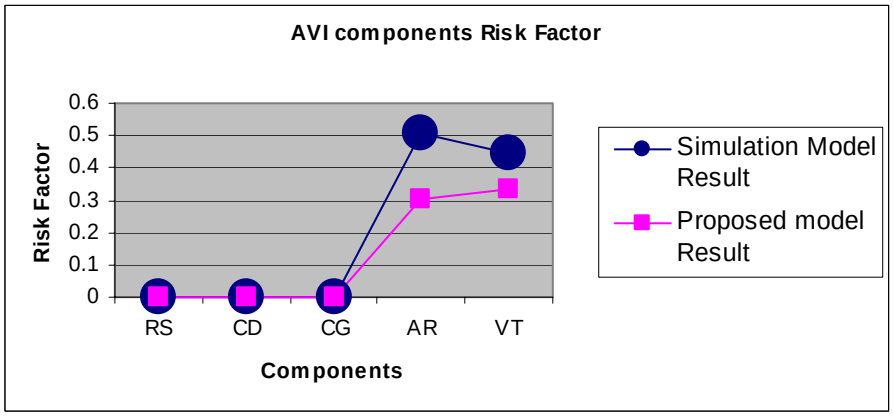



Figure 8.4 *AVI* scenario Components Risk Factor

***AAI scenario***

| Component | RS | CD | CG | AR | VT |
|---|---|---|---|---|---|
| Simulation Model Result | 0 | 0 | 0 | 0.95 | 0 |
| Proposed model Result | 0 | 0 | 0.00045 | 0.94905 | 0 |
| | | Correlation 1 | | | |

Table 8-8 *AAI* scenario Components Risk Factor

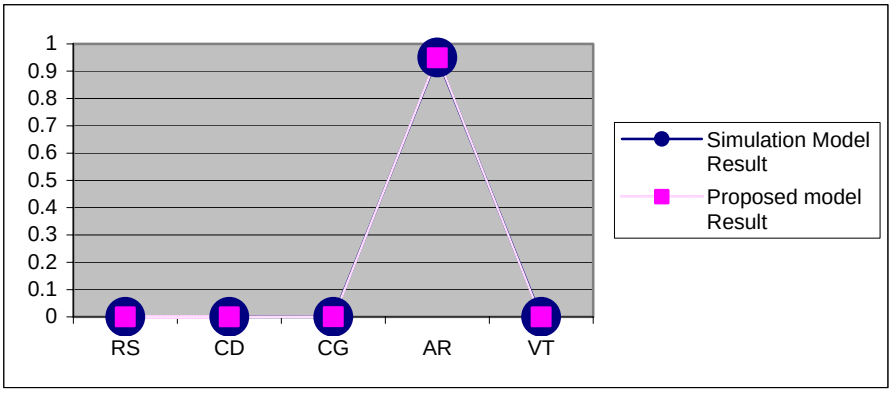



Figure 8.5 *AAI* scenario Components Risk Factor

***VVI scenario component risk factor***

| Component | RS | CD | CG | AR | VT |
|---|---|---|---|---|---|
| Simulation Model Result | 0 | 0 | 0 | 0 | 0.95 |
| Proposed model Result | 0 | 0 | 0.00045 | 0 | 0.94905 |
| | | | Correlation 1 | | |

Table 8-9 *VVI* scenario Components Risk Factor

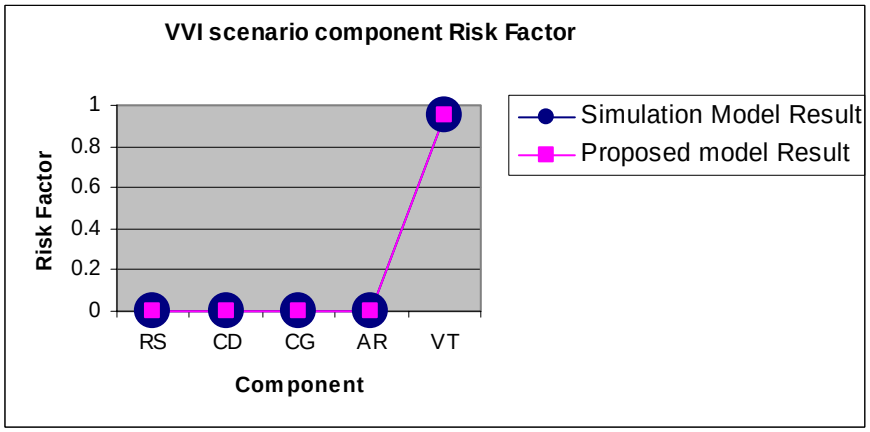



Figure 8.6 *VVI* scenario Components Risk Factor

***VVT scenario***

| Component | RS | CD | CG | AR | VT |
|---|---|---|---|---|---|
| Simulation Model Result | 0 | 0 | 0 | 0 | 0.95 |
| Proposed model Result | 0 | 0 | 0.00025 | 0 | 0.949 |
| | | Corrilation 0.999999974 | | | |

Table 8-10 *VVT* scenario Components Risk Factor

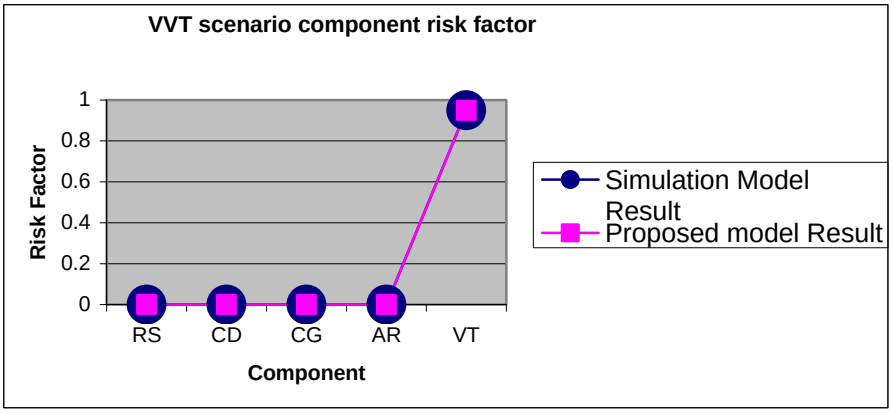



Figure 8.7 *VVT* scenario Components Risk Factor

***AAT Scenario***

| Component | RS | CD | CG | AR | VT |
|---|---|---|---|---|---|
| Simulation Model Result | 0 | 0 | 0 | 0.95 | 0 |
| Proposed model Result | 0 | 0 | 0.00025 | 0.949 | 0 |
| | | | Correlation 1 | | |

Table 8-11 *AAT* scenario Components Risk Factor

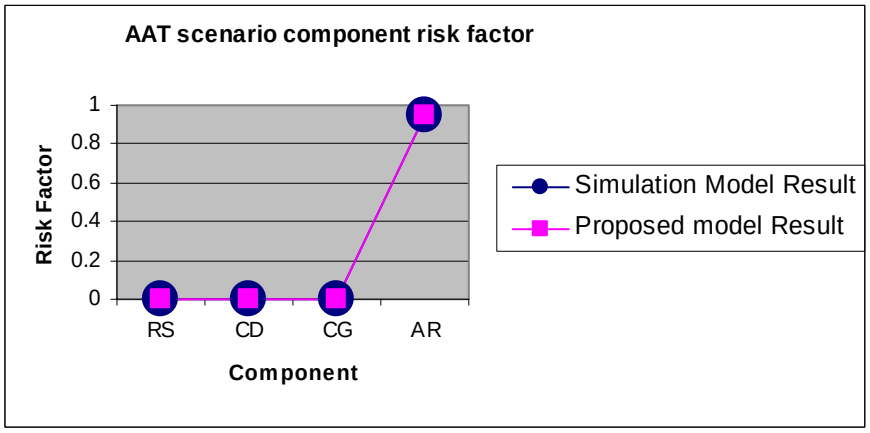



Figure 8.8 *AAT* Scenario Components Risk Factor

## 8.5 Severity methodology validation.

To validate this methodology we run fault injection experiment [Abdelmoez, 2004]. Fault injection experiment explores the behavior of the system in a case of failure resulting from an injected fault based on fault model.

The experiment shows that some failures in components/connectors considered in the domain expert fault tree are unrealistic. On the other hand, other unaccounted failures can be discovered in the experiment. Based on the results of the experment we can validate the proposed severity methodology.

We used a CCS case study (chapter 7) to apply the methodology and running experiment. We used the results estimated from the methodology (chapter 5) and from the experiment [Abdelmoez, 2004] with the previous validation criteria to validate the proposed severity methodology. We found some components/connectors considered to be contributing to certain failures by domain expert turned out not to have any role. Furthermore, Fault injection can discover some failures in components/ connectors that could not be discovered by the domain expert analysis.

### 8.5.1 Fault injection analysis

*This step has been presented here for the sake of completeness and the detailed implementation is presented in [Abdelmoez, 2004].*

In order to verify and validate the functionality of the command and control system, we use the environment, which consists of the system, the Fault-Generator and the Observer [Abdelmoez, 2004]. We put the system under investigation while faults being injected by using the Fault Generator (and the consequences are monitored through the Observer). The simulation is exercising all recoverable faults that the system can go through and react in the way specified in the requirements. We apply that fault injection methodology using Rose Real Time [Gomaa, 2000] simulation environment and log component states in order to observe the system response to the injected fault. Based on the consequences of resulting changes in component states on the system mission, we

analysis the log of the simulation to establish the fault tree for every component/connector with failures associated with the injected fault.

This analysis is being conducted for the system components and using various types faults according to the proposed fault model.

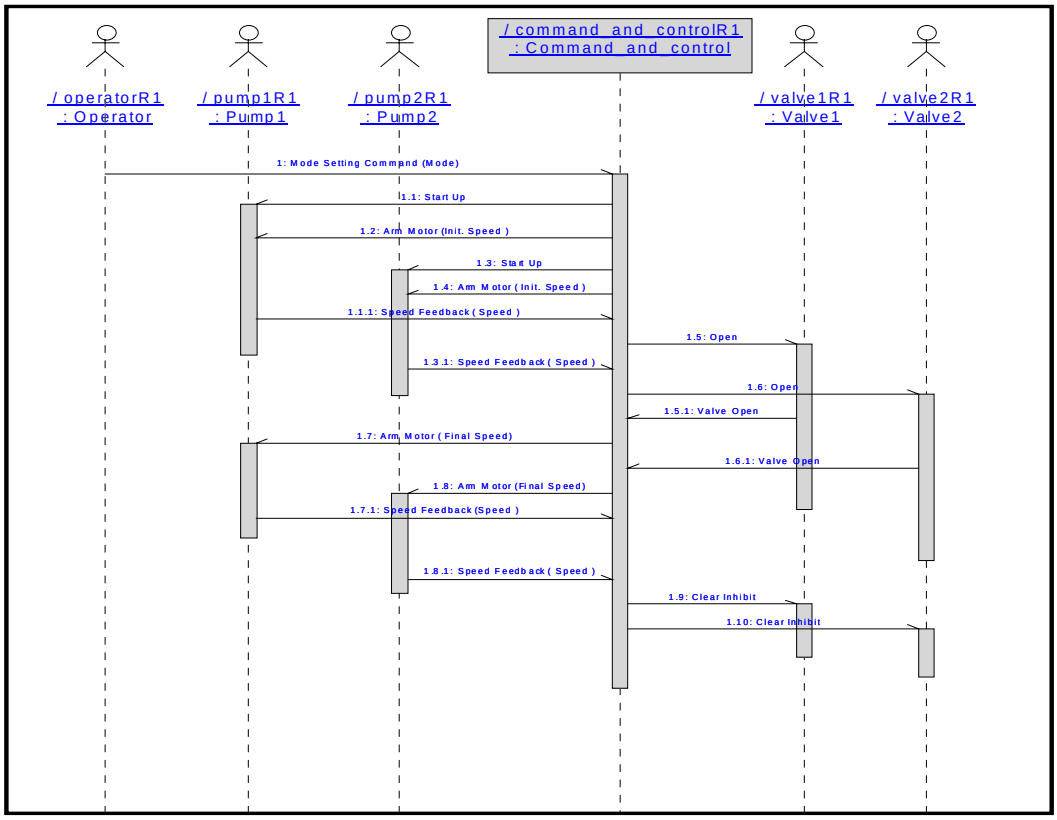



Figure 8.9 "*Mode setting*" system sequence diagram

Our objective is to come up with a study of the effects of failures on a component-by-component and a connector-by-connector basis. We inject faults (one at a time) into components and we run the simulator to study effects of a failure. Similarly, we inject faults (one at a time) into connectors and run the simulator to study the effects of a

failure. Figure 8.10 shows the fault tree of the external event "*Mode setting*" (Figure 8.9) which is a result of the step 3 of the proposed methodology. During this scenario (Figure 8.10) components ($C_1$ to $C_{10}$) interact to achieve the mode setting action. If we injected fault to that event ("*Mode Setting*") it will lead to system failure. This failure of the system mission could happen if the system received "*Mode setting*" in error. The resulting fault tree after injected fault is shown in Figure 4 without doted lines.

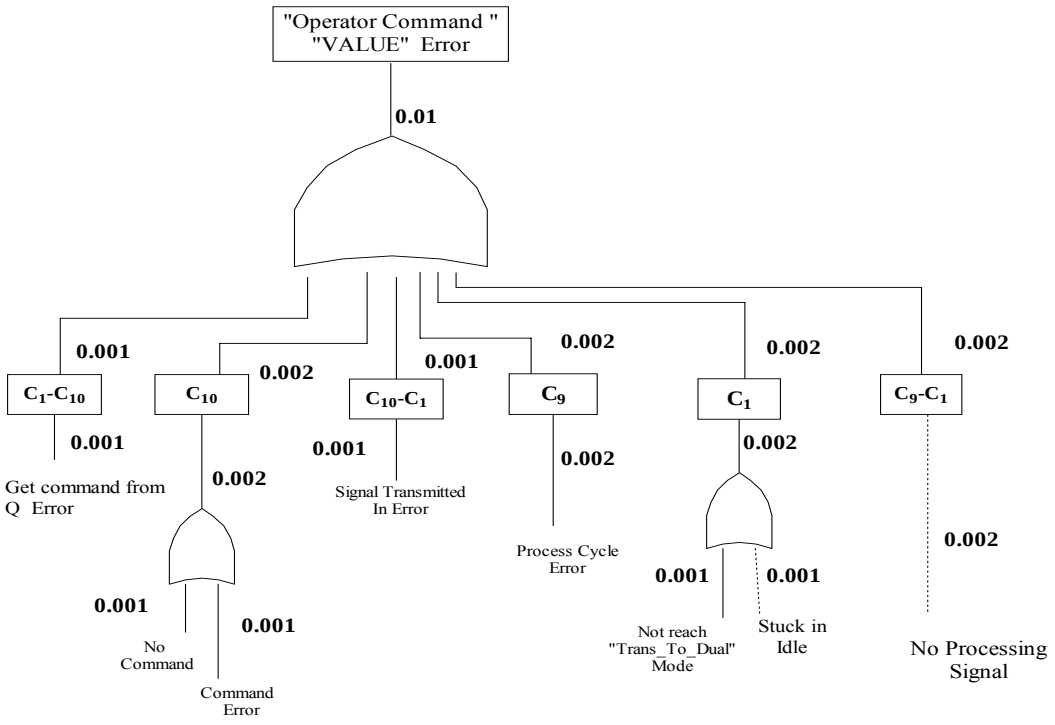



**Figure 8.10 "*Mode Setting*" Command" event fault tree**

As described above Figure 8.10 shows the fault tree resulting from the analysis (chapter 5 methodology step 3). The dotted lines are the branches of the tree for failures that the fault injection experiment showed to be unrealistic. Comparing the severity estimated based on the two fault tree structures, we could validate the severity assessment methodology as discussed in the following section.

### 8.5.2 Severity Methodology Validation

The fault tree output is used to estimate the cost of failure of any component or connector. We assume for every scenario that failure of any component or connector could lead to the failure of the system. By injecting fault and running the experiment we could estimate the exact contribution of components or connectors in the system failure during the execution of a specific scenario by means of exact fault tree (Figure 8.10 without doted lines). This will lead to a fault tree different from the one estimated from step 3 of the methodology (Figure 4) for every component/connector. This leads to different cost of failure than the one estimated from step 4 of the methodology. To evaluate the proposed severity methodology we use the previous criteria. In our validation process we assume that two variables, real severity (estimated based on the simulation model) and predicted severity (estimated from the proposed methodology), for all four severity categories with only two discrete values. Table 8-12 shows confusion matrix.

| | | Predicted Severity | | |
|---|---|---|---|---|
| | | Critical | Catastrophic | |
| Real Severity (Simulation results) | Critical | 16 | 1 | 17 |
| | Catastrophic | '0 | 18 | 18 |
| | | 16 | 19 | 35 |

Table 8-12 The confusion matrix for components of the CCS case study.

The proportion of correct predictions and the misclassifications are calculated as in the previous section.

***Proportion Correct***

$$A = \frac{n_{11} + n_{22}}{N} = (16+18)/35 = 0.97$$

***Misclassification rate***

$$f = \frac{n_{11}}{n_{11} + n_{12}} = 16/(16+1) = 0.94$$

The Type 1 misclassification rate is

$$1 - f = 1\text{-}0.94 = 0.06$$

Type 2 misclassification rate is $1 - S$ =

$$S = \frac{n_{22}}{n_{21} + n_{22}} = 18/(0+18) = 1$$

Type 2 misclassification rate = $1 - S$ = 1-1 = 0

The model works well for the prediction of catastrophic severity, and also for critical severity prediction, the model predict all the catastrophic severity but it predicts critical severity components by error factor 0.06 which is not bad.

**8.6 Conclusion**

As shown in this chapter we compared the result from the proposed methodology with the simulation results [Yacoub, 2003]. We conclude that the result of the proposed methodology gives high correlation for the component risk factor (scenario level or system level). The risk factor of connectors (scenario and system level) gives poor correlation.

After that we used three measures to evaluate the proposed risk technique. These measures are based on confusion matrix [Briand, 2000]. The risk model is considered as a good predictor for components risk factor but still need many case studies to draw a final conclusion. It is not considered to be a good predictor for a connector's risk factor and it needs many case studies before a final conclusion can be drawn.

# *Chapter 9*

## Architecture-level Risk Assessment Tool

### 9.1 Introduction

There is an increasing need for a tool that can be used to track the quality of a software product during the software design phase.

The product of the software development phases should be measurable in order to find and control its errors. Unfortunately, quality assurance methods with extensive automated support only apply in phases that are too late in the software life cycle to be really cost-effective. Many important product quality characteristics, such as performance, maintainability and risk assessment cannot be added late in the lifecycle, hence early warnings of poor quality expectation would be very useful to allow for early corrective measures. To automate and implement the proposed risk assessment methodology (chapter 6) we propose a prototype tool. We proceed in this chapter as follows:

Section 9.1 is the introduction; Section 9.2 presents a review, sections 9.3 and 9.4 present the proposed tool and its use case diagram, and finally, Section 9.5 is the conclusion and future work.

### 9.2 Review

Current research and development mostly goes in the direction of building tools to automate the software measurement process. Several tools of this type already exist.

For example, Lorenz and Kidd introduce a tool called OOMetric, which developed as software measurements tool for Smalltalk and VisualAge programs. The

basis for this tool can be found in [Lorenz, 1994]. Lorenz and Kidd describe measurements and give advice derived from a number of actual projects that have used object technology to deliver products. Reiner Dumke and his team [Dumke, 1996] developed a tool called ObjecTool, which is used for analyzing C++ programs.

Fenelon et al [Fenelon, 1999] have developed a tool support for the estimation of software quality. El-Emam [El-Emam, 2001c], presents a metrics analyzer tool for the C++ source code. This analyzer collects its data from C++ code. These data are analyzed to calculate a set of design metrics. In [Nenonen, 2000], the author presented a tool for measuring static metrics from UML diagrams. These mentioned tools are highly specialized in the approaches they implement and the particular phase of the software life cycle in which they are applicable.

Many of these tools are source code metrics based decision making tools (language oriented tools). None of the mentioned tools deled with any kind of dynamic metrics of the software.

Source code metrics are affected by the programming style of the programmer. Th programming language itself with its structures affects the metrics results. When calculating the metrics from architectural descriptions like UML, we achieve independence of languages and human factors [Hitz, 1998].

On the other hand, some tools [Nenonen, 2000] do get a description from intermediate file by using certain CASE tools; they can be used in the design phase as well, but they only produce static metrics to describe the model with limited capability, which is not enough to accurately represent the dynamic behavior of the architecture.

They even require the software model to be in a specific chosen format, which is not convenient for popular use.

Our proposed tool is a dynamic metrics based tool, UML based tool, and used early on in the design phase for the prediction of software architectural element risk.

## 9.3 Proposed Tool

*The implementation and the coding of the tool is presented in details in [Wang, 2003].*

To obtain useful quality assurance information early enough for improvement purposes, we base our quality predictions on measurements and calculations of the high-level design diagrams obtained from UML artifacts. To automate and implement the risk assessment methodology we propose an automated UML-based software risk assessment prototype tool.

The proposed prototype tool is called **A**rchitecture-level **R**isk **A**ssessment **T**ool (ARAT), and is used to demonstrate the process of risk assessment. ARAT is built to implement the risk assessment methodology (presented in chapter 6), by manipulating the data acquired from domain expert and measures obtained from UML artifacts.

ARAT measures dynamic metrics proposed in [Hassan, 2001] (chapter 4) and automatically analyzes the quality of the architecture to produce architectural level software risk assessment [Katerina, 2003]. Figure 9.1 depicts the overall architecture of ARAT.

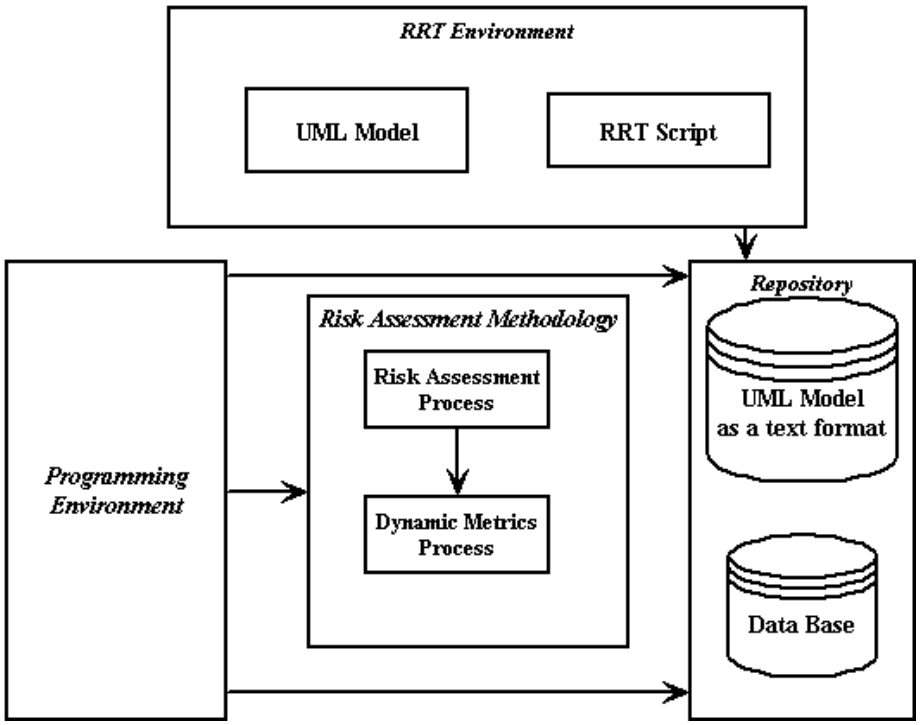



Figure 9.1 Overall architecture of ARAT

Rational Rose is a well established UML modeling tool in software engineering [Douglass, 2000] To model the software under study we used Rose RT as a modeling environment. RRT is specifically useful for modeling real time system [ ].

To be able to quantify measurements from the visual UML model, we used RRT script to convert the UML visual model to readable text format. The script automatically goes through all the diagrams of the target software UML model. It captures the detailed information of the visual UML model and stores it in the repository for further analysis and calculation.

On the top of these information of UML model we use java as a programming environment with the proposed risk methodology [Katerina, 2003] and severity analysis process [Hassan, 2003c] to come up with detailed risk assessment of the software model. For historical review and comparing multiple versions of the design we store all the

information of the model and the results of risk assessment process on a repository as shown in figure 9.1.

### 9.3.1 ARAT Use Case

In this section we will describe the use case diagram of ARAT, which is shown in Figure 9.2.

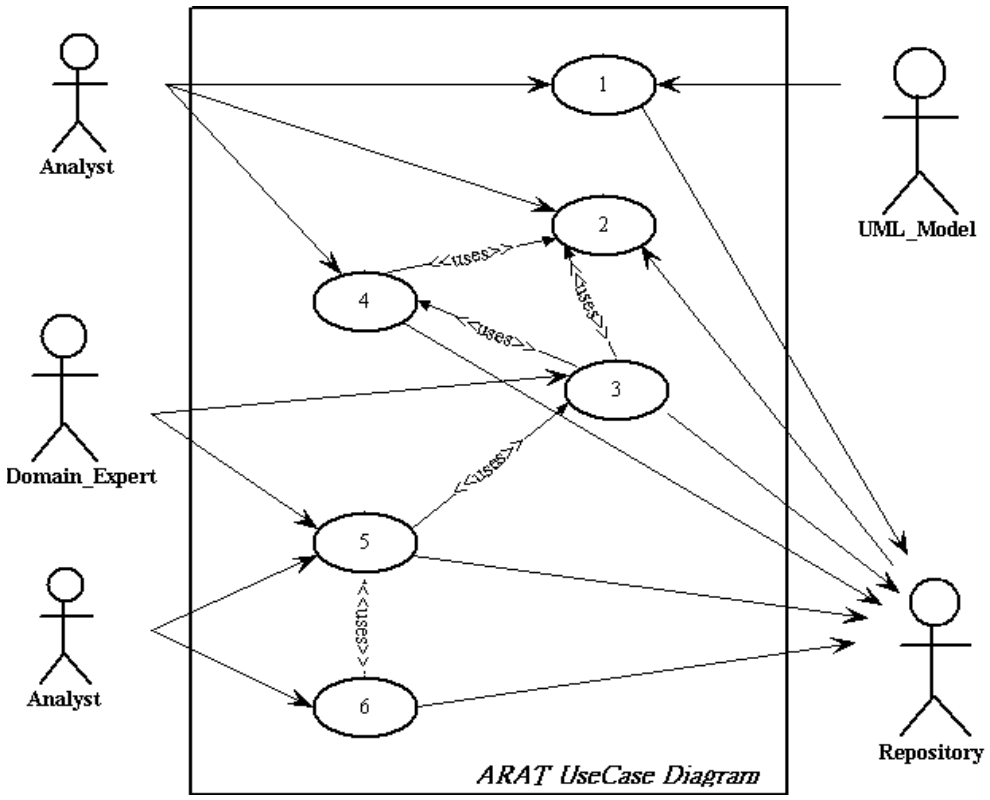



Figure 9.2 ARAT Use Case diagram

***Collection model information (use case 1)***

The implementation of this part of the tool is done using the RRT script with the UML model under RRT environment. The input of this module is the UML visual model and the output is the detailed information of the UML model as a text format. Figure 9.3 shows an example of the output for a given input (case study shown in Figure 7.3).

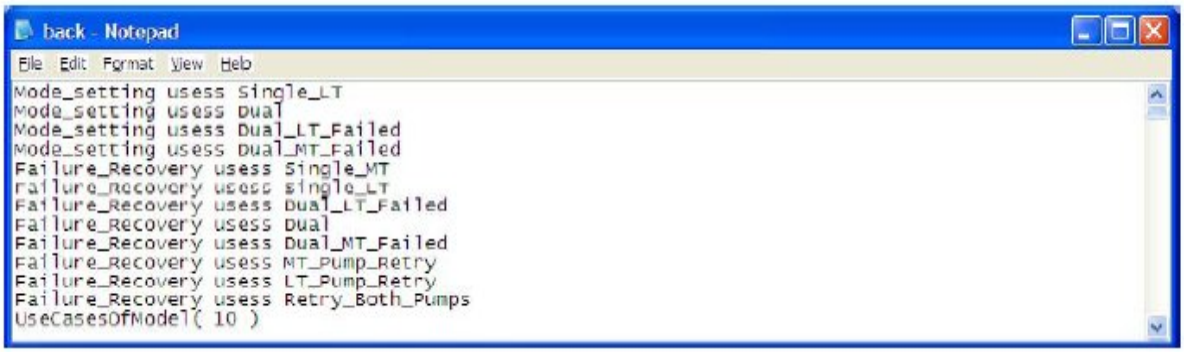



Figure 9.3 Information captured from use case diagram pacemaker

***Retrieve model information (use case 2)***

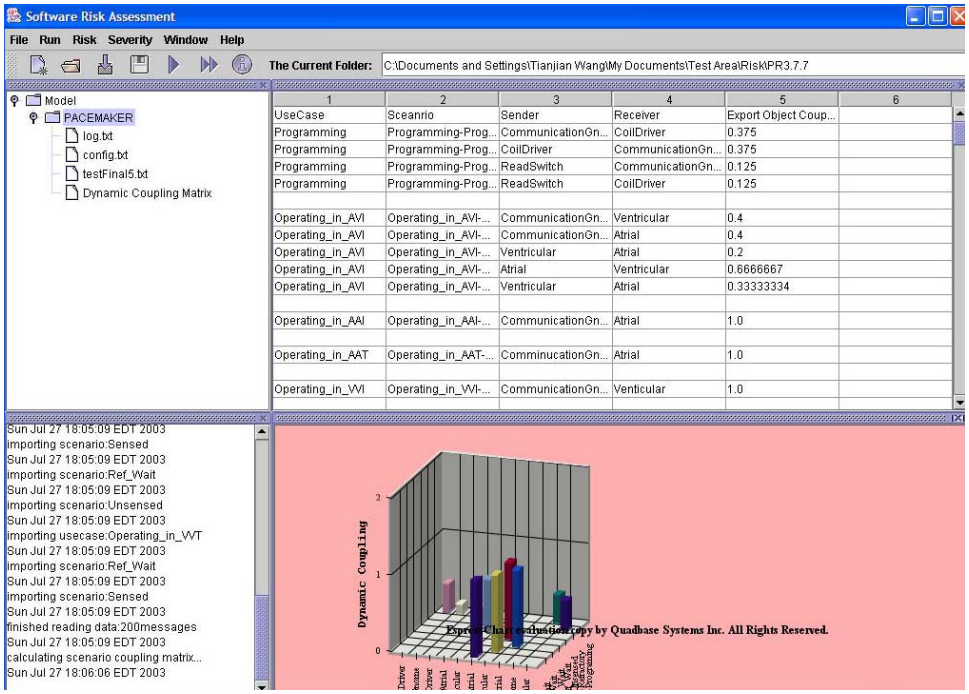
This use case is used for input/output the information to/from the repository. This information could be about the UML model, dynamic metrics, severity, or risk assessment.

***Estimate dynamic metrics (use case 3)***

This part of the tool is the implementation of dynamic metrics presented in chapter 4. Figure 9.4 shows example of coupling for connectors for pace maker case study.

***Estimate Component/Connector risk factor (use case 4)***

This part of the tool used with input from severity assessment (chapter 5), and component/connector probability of failure (chapter 4) to estimate the risk factor for each component/connector.




| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| UseCase | Sceanrio | Sender | Receiver | Export Object Coup... | |
| Programming | Programming-Prog... | CommunicationGn... | CoilDriver | 0.375 | |
| Programming | Programming-Prog... | CoilDriver | CommunicationGn... | 0.375 | |
| Programming | Programming-Prog... | ReadSwitch | CommunicationGn... | 0.125 | |
| Programming | Programming-Prog... | ReadSwitch | CoilDriver | 0.125 | |
| | | | | | |
| Operating_in_AVI | Operating_in_AVI-... | CommunicationGn... | Ventricular | 0.4 | |
| Operating_in_AVI | Operating_in_AVI-... | CommunicationGn... | Atrial | 0.4 | |
| Operating_in_AVI | Operating_in_AVI-... | Ventricular | Atrial | 0.2 | |
| Operating_in_AVI | Operating_in_AVI-... | Atrial | Ventricular | 0.6666667 | |
| Operating_in_AVI | Operating_in_AVI-... | Ventricular | Atrial | 0.33333334 | |
| | | | | | |
| Operating_in_AAI | Operating_in_AAI-... | CommunicationGn... | Atrial | 1.0 | |
| | | | | | |
| Operating_in_AAT | Operating_in_AAT-... | CominucationGn... | Atrial | 1.0 | |
| | | | | | |
| Operating_in_VVI | Operating_in_VVI-... | CommunicationGn... | Venticular | 1.0 | |

Figure 9.4 connector coupling for every scenario of pacemaker

***Estimate scenario level risk actor (use case 5)***

To implement the estimation of scenario level risk factor we use this part of the tool. The estimation of scenario level risk factor is shown in chapter 6. The detailed analysis of the scenario level risk factor is presented in [Ajith, 2004]

***Estimate system level risk factor (use case 6)***

This part of the tool is the implementation of system level risk factor described in chapter 6. The detailed description and design of ARAT is presented in [Wang, 2003].

Figure 9.5 shows the GUI of the ARAT tool.

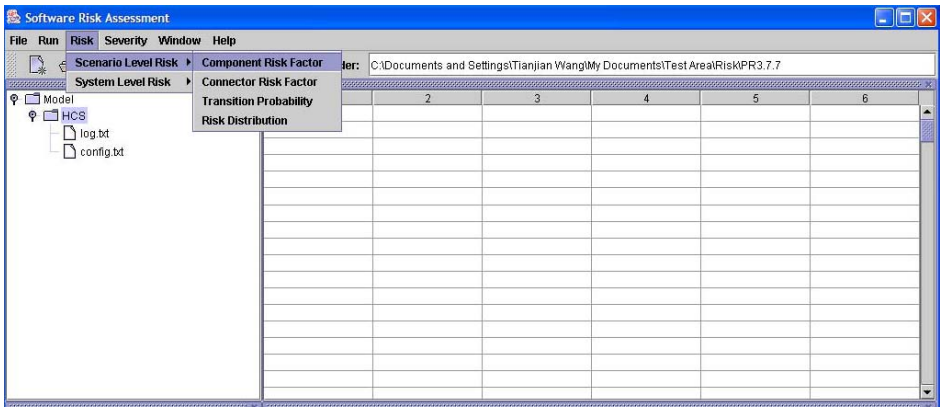



Figure 9.5 is one of the examples of console GUI from the ARAT

## 9.4 Conclusion

Our proposed architectural risk assessment tool is designed to be utilized during the early software development phases. We use Rational Rose Real Time (RRT) as a modeling environment [Rose, 2001] We use Rose Script for collecting the UML model information from the UML visual model. This tool support of the risk assessment methodology could help software managers and engineers to control and optimize the software development process.

The main benefit of the proposed tool is that it enables the automation of early assessment of system risk at the architectural level and hence makes it possible for the analyst to identify critical components and connectors early in the software lifecycle.

In summary, the main objectives of this tool are listed below:

1. ARAT could carry on the risk assessment as early as the design phase in the lifecycle of the software development.
2. By using ARART, we would be able to precisely compute the component/connector risk factor, scenarios risk factor, use case or system risk factors.

3. the manager will be able to determine the distribution of the scenario, use case or system risk factor over different severity classes.
4. With ARAT, the analyst will be able to identify critical components based on the risk estimation.
5. ARAT is flexible enough to include more functional modules that will be developed in the future, like performance analysis module, hazard analysis module etc.

In the future, we intend to integrate severity assessment module to ARAT to automate the severity analysis methodology proposed in chapter 5.

# *Chapter 10*

## Conclusion and Future work

Several conclusions emerge from this research. We conclude that this work is a promising and significant step in meeting our research objectives: to develop risk assessment methodology based on measurable parameters that could be automatically collected and analyzed in the early software design phase based on UML artifacts. And to develop a severity assessment methodology based on UML artifacts. Several other conclusions are drawn with respect to our aims of the research.

This research presents the architectural-level reliability-based risk assessment methodology. In this research we propose a methodology for risk assessment based on the UML specifications such as use cases and sequence diagrams that can be used in the early phases of software life cycle. The proposed methodology uses dynamic complexity and dynamic coupling metrics that are obtained from the UML specifications [Hassan, 2001].

The risk assessment methodology presented in this research [Katerina, 2003] considers both component and connector risk factors. It is used for calculating the risk factors of various components and connectors and estimating a risk factor of scenarios use cases and system level. We combine the probability of software architectural element failure with the severity of that failure to estimate the risk factor of software architectural element.

To estimate the probability of software architectural element failure we propose dynamic metrics (dynamic complexity/dynamic coupling) for the software architectural element. The proposed metrics could be obtained at early development phases from UML models.

To estimate the severity of software failure we propose a UML based severity methodology (chapter 5). This methodology describes a process for estimating severity of each software architectural element component/connector at the software architectural level as well as severity of system scenarios [Hassan, 2003]. The process is based on dynamic UML specifications, taking into account the possibility of component/connector cost of failures [Hassan, 2003c]. The severity methodology is based on hazard analysis techniques [Hassan, 2005]. FFA is used as a top down approach based on system scenarios to identify the system level failures, FMEA is used as a bottom up appraoch based on the detailed view of the system to identify the possible causes component/connector failures and FTA correlates the results of FMEA and FFA.

Since the risk assessment methodology is entirely analytical and provides a closed form solution, it is very suitable for automation. In fact, a prototype of the risk assessment tool written in JAVA [Wang, 2003] which reads the embedded UML information from Rational Rose visual model, and calculates the various risk factors has already been developed.

In summary the proposed methodology is an efficient method to estimate risk factors on different levels of software design and we could estimate overall system risk factor. It enables us to estimate scenarios and use cases risk factors which enable us to

focus on the high-risk scenarios and uses cases even though they may be rarely used and therefore may not contribute significantly to the overall system risk factor.

Next, we estimate the distribution of the scenarios/use cases/system risk factors over different severity classes, which allow us to make a list of critical scenarios in each use case, as well as a list of critical use cases in the system for every severity class. Finally, we identify a list of critical components and connectors that has high risk values in high severity classes.

Our future work is focused on generalization of the methodology presented in this dissertation. Thus, we will extend this methodology to relax the assumption of independent use cases. We will consider different kinds of dependencies that might be present in the UML use case diagrams (i.e. considers the various relationships between the use cases) and the way to derive their risk factors. Another direction of our future research is the development of performance based risk assessment methodology [Cortellessa, 2005] and requirement based risk assessment methodology [Appukutty, 2005].

In the future, we could also do the following:

- Empirically validate the proposed dynamic metrics and their correlation with design quality attributes,
- Develop severity assessment module to be integrated with ARAT to automate the severity analysis methodology proposed in chapter 5,
- Extend the risk methodology for other kinds of risk (Ex. maintainability risk, performance risk, requirement risk),

- Validate the severity methodology using fault injection experiment using many case studies.
- Apply the mitigation action required as a result of risk assessment on a specific case study (ex. redesign the system) and reassess the risk.
- Extend the system to handle the risk management not only risk assessment (based on risk assessment identify and study the alternatives required for this software product).

# *References*


[Abdelmoez, 2004] W. Abdelmoez, D.M. Nassar, M. Shereshevsky, N. Gradetsky, R. Gunnalan, H.H. Ammar, Bo Yu, A. Mili, "Error Propagation In Software Architectures", *Proc. 10th IEEE International Software Metrics Symposium (METRICS 2004)*, Chicago, IL, Sept. 2004.

[Abreu, 1996] Fernando Abreu, Walcélio Melo,"Evaluating the Impact of Object-Oriented Design on Software Quality***",*** *IEEE Proceedings of the 3rd International Software Metrics Symposium (METRICS'96),* Berlin, Germany, March 1996.

[Ajith, 2004] Ajith Reddy Guedem, "Software Architectural Risk Assessment", Master Thesis submitted to the College of Engineering and Mineral Resources at West Virginia University, 2004.

[Allenby, 2001] Karen Allenby, Tim Kelly, "Deriving Safety Requirements Using Scenarios", *5th IEEE Int'l Symposium on Requirements Engineering*, August 27-31, 2001, 228-236, Toronto, Canada.

[Allenby, 2001] Karen Allenby, Tim Kelly, "Deriving Safety Requirements Using Scenarios", *5th IEEE Int'l Symposium on Requirements Engineering*, August 27-31, 2001, 228-236, Toronto, Canada.

[Almeida, 1998] M. Almeida, H. Lounis, and W. Melo, "An Investigation on the Use of Machine Learned Models for Estimating Correction Costs", *Proceedings of the 20th International Conference on Software Engineering*, pp. 473-476, 1998.

[Ammar, 1997] H. Ammar, T. Nikzadeh, and J. Dugan, "A Methodology for Risk Assessment of Functional Specification of Software Systems Using Coherent Petri

Nets", *Proc. 4th Int'l Software Metrics Symp.* (Metrics'97), Albuquerque, New Mexico, 1997, pp. 108-117.

[Appukutty, 2005] K.Appukutty, Hany H. Ammar, Katerina Goseva Popstajanova, "Software Requirement Risk Assessment Using UML", *AICCSA* January ' 05.

[Arron, 1998] Aaron B. Binkley and Stephen R. Schach, "Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures", *International Conference on Software Engineering*, pp 452-455, 1998.

[Arron, 1998] Aaron B. Binkley and Stephen R. Schach, "Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures", *International Conference on Software Engineering*, pp 452-455, 1998.

[Brockers, 1995] Alfred Bröckers, "Process-based software risk assessment" *In Proceedings of EWSPT4*, Noordwijkerhout, The Netherlands, 1995.

[Briand, 1993] Briand, L. C., Thomas, W. M., and Hetmanski, C. J., Modeling and managing risk early in software development", *Proc. 15th Int. Conf. Software Eng.*, 55-65, 1993.

[Briand, 1993a] L. Briand, V. Basili, and C. Hetmanski, "Developing Interpretable Models with Optimized Set Reduction for Identifying High-Risk Software Components," *IEEE Transactions on Software Engineering*, vol. 19, no. 11, pp. 1028-1044, 1993.

[Briand, 2000] L. Briand, J. Wüst, John W. Daly, V. Porter, "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems", *Journal of Systems and Software*, 51 (2000), pp. 245-273.

[Bach, 1999] James Bach, "Heuristic Risk-Based Testing", *Software Testing and Quality Engineering Magazine*, Nov., 1999.

[Bass, 2003] Len Bass, Paul Clements, Rick Kazman, "Software Architecture in Practice", 2/E, Addison Wesley Professional, Copyright: 2003.

[Benlarbi, 1999] S. Benlarbi and W. Melo, "Polymorphism Measures for Early Risk Prediction", *21st Intl Conference on Software Engineering*, 1999, PP 334-344.

[Benlarbi, 1999] S. Benlarbi and W. Melo: "Polymorphism Measures for Early Risk Prediction", *Proceedings of the 21st International Conference on software Engineering*, pp 334-344, 1999.

[Booch, 1999] Grady Booch, James Rumbaugh and Ivar Jacobson, *The Unified Modeling Language User Guide* (Addison-Wesley, 1999) ISBN 0-201-57168-4.

[Briand, 1998] L. Briand, J. Wuest, S. Ikonomovski, and H. Lounis "A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: An Industrial Case Study", *IEEE 21st International Conference on Software Engineering (ICSE'99*), Los Angeles, USA, May 1999.

[Briand, 2000] Lionel C. Briand, Walcelio L. Melo, Jürgen Wüst, "Assessing the Applicability of Fault-Proneness Models Across Object-Oriented software Projects", *IEEE Transactions on Software Engineering*, pp 706-720, 2002.

[Cheung, 1980] R. C. Cheung, "A User – Oriented Software Reliability Model", *IEEE Trans. Software Eng,* vol.6, no.2, 1980, pp. 118-125.

[Cheung, 1980] R.C. Cheung, "A User-Oriented Software Reliability Model", *IEEE Trans. Software Eng*., vol. 6, no. 2, pp. 118-125, 1980.

[Carpenter, 1999] Paul B. Carpenter,"Verification of Requirements For Safety-Critical Software", *SIGAda '99 Annual Intl Conference: The Engineering of Industrial Strength REAL-TIME Software & Distributed Systems Using Ada and Related Technologies,* 1999

[Charles, 1996] Charles F. Eubanks, S. Kmenta, and K. Ishii, "System Behavior Modeling as a Basis for Advanced Failure Modes and Effects Analysis," *ASME Computers In Engineering Conference*, Sept. 1996, Irvine, CA.

[Charles, 1997] Charles F. Eubanks, Steven Kmenta, Kosuke Ishii, "Advanced Failure Modes and Effects Analysis Using Behavior Modeling", *Proceedings of DETC'97, 1997 ASME Design Engineering Technical Conferences and Design Theory and Methodology Conference*, September 14-17, 1997, Sacramento, California.

[Chidamber, 1994] Chidamber, S.R., and C.F. Kemerer, "A metrics suite for object-oriented Design", IEEE TSE, 20(6) June 1994, pp. 476-493.

[Cortellessal, 2005] V. Cortellessa1, K. Goseva-Popstojanova, K. Appukutty, A. Guedem, A. Hassan, R. Elnaggar, W. Abdelmoez, and H. H. Ammar, "Performance-based Risk Analysis of UML Models", *submitted to IEEE Transaction of Software Engineering (TSE).*

[DiMarco, 1995] DiMarco, P., C. Eubanks and K. Ishii, "Service Modes and Effect Analysis: Integration of Failure Analysis and Serviceability Design," *Proc. of the 15th Annual International Computers in Engineering Conference (CIE '95),* Boston, MA, pp. 833-840.

[DoD, 1997] Report No. MIL-STD-882D U.S. Department of Defence.

[Douglass, 2000] B. P. Douglass, "Real-Time UML: Developing Efficient Objects for Embedded Systems", 2nd Edition, Published by Addison Wesley, 2000.

[Dumke, 1996] Reiner R. Dumke, Erik Foltin, R. Koeppe, and A.S. Winkler, „Measurement - based Object-oriented Software Development of the Software Project" Technical Report Preprint No. 6, Otto-von-Guericke-University at von Magdeburg, Software Measurement Laboratory 1996.

[El-Emam, 1999c] Khaled El-Emam, S. Benlarbi, and N. Goel, "Comparing Case-based Reasoning Classifiers for Predicting High Risk Software Components", *Technical Report, NRC/ERB-1058*, September 1999. NRC 43602.

[Elemam, 1999z] K. El Emam and W. Melo , "The Prediction of Faulty Classes Using Object - Oriented Design Metrics", *tech. report, NRC 43609*, National Research Concil Canada, Institute for Information Technology, 1999.

[Elemam, 2001b] K. El-Emam, S. Benlarbi, N. Goel, and S. Rai, "Comparing Case-Based Reasoning Classifiers for Predicting High Risk Software Components," *Journal of Systems and Software (to appear)*, 2001.

[Elemam,1999f] Khaled El Emam, Walcelio Melo, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics", *NRC/ERB-1064. National Research Council* Canada, Institute for Information Technology, November 1999.

[Fenton, 1996] Fenton N., Pfleeger, S., "Software metrics: A Rigorous Approach", Thomson Publisher, 1996.

[Fenton, 1999b] Fenton, N., Pfleeger S., Glass R, " Science and Substance: A Challenge to Software Engineers", *IEEE Software*, vol. 11(4), pp. 86-95, 1999.

[Fenton, 1999mm] Norman E Fenton Niclas Ohlsson," Quantitative Analysis of Faults and Failures in a Complex Software System*", IEEE Transactions on Software Engineering,* pp 794-814, 2000.

[Fenton, 2000] Norman E. Fenton and Martin Neil "Software metrics: roadmap", *International Conference on Software Engineering, Proceedings of the 22nd nternational Conference on The future of Software engineering* 2000 June 4 - 11, 2000, Limerick Ireland, Pages 357-370 .

[Gilchrist, 1993] Gilchrist, W. "Modeling failure modes and effects", *International Journal of Quality and Reliability Management,* 16-23, 10(5), 1993.

[Gordis, 1996] L. Gordis, "*Epidemiology*", W. B. Saunders Company, 1996.

[Gomaa, 2000] H. Gomaa, *Designing Concurrent, Distributed, and Real-Time Applications with UML*. Addison-Wesley, 2000.

[Genero, 2003] Marcela G., Mario P., "Building UML Class Diagram Maintainability Prediction Models Based on Early Metrics", *IEEE 9th International Software Metrics Symposium (METRICS'03)* September 03 - 05, 2003, Sydney, Australia.

[Harrison, 2000] R. Harrison, S. Counsell, R. Nithi "Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems**",** *Journal of Systems and Software*, V 52 , No 2-3, June 2000, PP 173 - 179 Special issue on Evaluation and assessment in software engineering.

[Hassan, 2003] A. Hassan, K. Goseva-Popstojanova, H. Ammar , "Methodology for Architecture Level Hazard Analysis, A Survey", ACS/IEEE Intl. Conference on Computer Systems and Applications (AICCSA'03), Tunis, Tunisia, July 14-18, 2003.

[Hassan, 2003C] A. Hassan, W. Abdelmoez , A. Guedem, K. Apputkutty, K. Goseva-Popstojanova, H. Ammar, "Severity Analysis at Architectural Level Based on UML

Diagrams", Proceedings of the 21st Int'l System Safety Conference, Ottawa, Canda, Aug. 2003, PP 571-580.

[Hassan, 2005] Ahmed Hassan, Katerina Goseva – Popstojanova, Hany Ammar,"A methodology for Severity Analysis of Software Architectures", *The 51st Annual Reliability & Maintainability Symposium (RAMS),* 2005

[Heemstra, 2003] Fred J. Heemstra, Rob J. Kusters, Huibert de Man, "Guidelines for Managing Bias in Project Risk Assessment*", International Symposium on Empirical Software Engineering (ISESE'03)*, September 30 - October 01, p. 272 – 281,2003, Roman Castles (Rome), Italy

[Hitz, 1998] M. Hitz, K. Neuhold, "A Framework for Product Analysis", *OOPSLA 1998, Workshop on Model Engineering*, *Methods and Tools Interaction with CDIF*, 1998.

[Hassan, 2001] A. Hassan, W. Abdelmoez, R. Elnaggar, and H. Ammar, "An Approach to Measure the Quality of Software Designs from UML Specifications", *7th Int'l Conf. Information Systems, Analysis and Synthesis*, July. 2001, Vol. IV, pp 559-564.

[IEEE, 1993] IEEE Computer Society : IEEE Standard for a software Quality Metrics Methodology, IEEE Standard 1061,May-1998

[Johannessen, 2001] Per Johannessen, Christian Grante,Anders Alminger, Ulrik Eklund Jan Torin, "Hazard Analysis in Object Oriented Design of Dependable Systems", *Proceeding of the 2001 Int'l conference on Dependable Systems and Networks,* 2001, 507-512, Goteborg, Sweden.

[Kamiva, 1999] Toshihiro Kamiya, Shinji Kusumoto and Katsuro Inoue, "Prediction of Fault-proneness at Early Phase in Object-Oriented Development", *2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing,* May 02 - 05, 1999, Saint-Malo, France.

[Kemnta, 2000] Steven Kmenta, “Scenario- Based FMEA: A Life Cycle Cost Perspective”, *Proceedings of DETC 2000, ASME Design Engineering Technical Conferences,* September 10 - 14, 2000, Baltimore, Maryland.

[Khoshgoftaar, 1993] Khoshgoftaar, T. M., J. C. Munson, and D.L. Lanning,"Dynamic System Complexity", *Proc. of International Software Metrics Symposium, Metrics'93*, Baltimore MD., May 1993, pp129-140.

[Khoshgoftaar, 1999] T. Khoshgoftaar, E. Allen, W. Jones, and J. Hudepohl, "Classification Tree Models of Software Quality Over Multiple Releases," *Proceedings of the International Symposium on Software Reliability Engineering*, pp. 116-125, 1999.

[Katerina, 2001] K. Goseva – Popstojanova and K. S. Trivedi, “Architecture Based Approach to Reliability Assessment of Software Systems”, *Performance Evaluation*, vol. 45, no. 2-3, June 2001, pp. 179-204.

[Khoshgoftaar, 1995a] T. Khoshgoftaar, E. Allen, K. Kalaichelvan, N. Goel, J. Hudepohl, and J. Mayrand, "Detection of Fault-Prone Program Modules in a Very Large Telecommunications System", *Proceedings of the 6th International Symposium on Software Reliability Engineering*, pp. 24-33, 1995.

[Khoshgoftaar, 1996b] Khoshgoftaar, T., Allen, E., Bullard, L., Halstead, R., Trio, G., A., “Tree Based Classification Model for Analysis of a Military Software System”, *Proc. of the IEEE High-Assurance Systems Engineering Workshop*, 244-251, 1996b.

[Khoshgoftaar, 1997] T. Khoshgoftaar, K. Ganesan, E. Allen, F. Ross, R. Munikoti, N. Goel, and A. Nandi: “Predicting Fault-Prone Modules with Case-Based Reasoning”,

*Proceedings of the Eighth International Symposium on Software Reliability Engineering*, pages 27-35, 1997.

[Khoshgoftaar, 1999] Khoshgoftaar, T., Allen, E., Jones, W., Hudepohl, J., "Classification Tree Models of Software Quality Over Multiple Releases", *Proc. of the International Symposium on Software Reliability Engineering*. 1999

[Kirsten, 1998] Kirsten M. Hansen, Anders P. Ravn, Victoria Stavridou, "From Safety Analysis to Software Requirements", *IEEE Transaction on Software Engineering* 24, No. 7, July 1998, 573-584.

[Knight, 2000] John C. Knight**,** "Software Challenges in Aviation Systems", *International Conference of Computer Safety, Reliability and Security*, September 2002, Catania, Italy.

[Lonchamp, 1993] Jaques Lonchamp. A structured conceptual and terminological framework for software process engineering. In *Proceedings of the Second International Conference on the Software Process*, pages 41–53. IEEE Computer Society Press, February 1993.

[Letha, 2000 ] Letha Etzkorn Harry Delugach, " Towards a Semantic Metrics Suite for Object-Oriented Design", *IEEE Conference*, *Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*, July 30 – Aug. 03, 2000, Santa Barbara, California.

[Li, 1993] Li, W., and S. Henry, "Object Oriented Metrics that predict Maintainability", *Journal of Systems and Software*, V23, No 2, pp 111-122, 1993.

[Lindsay, 1997] P. A. Lindsay and J. A. McDermid. “A systematic approach to software safety integrity levels”, *Proceedings of 16th International Conference on Computer Safety, Reliability and Security (SAFECOMP'97)*, 1997, 70-82.

[Liu, 2000] Shaoying Liu, “Verifying Formal Specifications Using Fault Tree Analysis”, International Symposium on Principles of Software Evolution, Nov. 1-2, 2000, Kanazawa, Japan.

[Lanubile, 1997] F. Lanubile and G. Visaggio, "Evaluating Predictive Quality Models Derived from Software Measures: Lessons Learned", *Journal of Systems and Software*, vol. 38, pp. 225-234, 1997.

[Lorenz, 1994] Mark Lorenz and Jeff Kidd, “Object-Oriented Software Metrics”, Object-Oriented Series, Printice Hall, Engelwood Clifffs, NJ, USA, 1994.

[Marcela, 2003] Marcela Genero, Mario Piattini, Esperanza Manso, Giovanni Cantone, “

[McCabe, 1976] McCabe, T., "A Complexity Metrics", *IEEE Trans on Software Engineering,* Vol 2, No. 4, Dec 1976, pp308-320

[McDermid 2000] J. A. McDermid and David J Pumfrey, “Assessing the Safety of Integrity Level Partitioning in Software”, *Proc. Of the 8th Safety-Critical Systems Symposium*, Southampton, UK, pp. 134-152, 2000.

[McDermid, 1995] McDermid, J.A., Nicholson, M., Pumfrey, D.J. & Fenelon, P., “Experience with the application of HAZOP to computer-based systems”, *IEEE, COMPASS '95: Proceedings of the Tenth Annual Conference on Computer Assurance*, pp. 37-48, 1995, Gaithersburg, MD.

[Mei,1999] Mei-Huei Tang Ming-Hung Kao Mei-Hwa Chen, "An Empirical Study on Object-Oriented Metrics", *6th IEEE International Symposium on Software Metrics*, November 04 - 06, 1999, Boca Raton, Florida.

[MIL_STD, 1984] US Military Standard, "Procedures for Performing Failure Mode Effects and Criticality Analysis", *US MIL_STD_1629A / Notice 2*, Nov. 1984.

[MIL_STD, 1984] US Military Standard, "Procedures for Performing Failure Mode Effects and Criticality Analysis", US MIL_STD_1629A / Notice 2, Nov. 1984

[Moser, 1997] Simon Moser, Vojislav B. Misic, "Measuring Class Coupling and Cohesion: A Formal Metamodel approach", *IEEE 4th Asia-Pacific Software Engineering and International computer Science Conference (APSEC' 97/ ICSC '97,* December 1997.

[Musa, 1993] J. D. Musa, "Operational profiles in software reliability engineering", *IEEE Software*, 10(2):14–32, Mar. 1993.

[Musa, 1996]J. Musa, G. Fuoco, N. Irving, D. Krop°, and B. Juhlin, "The Operational Profile", Handbook of Software Reliability Eng., M. Lyu, ed., pp. 167-216, 1996.

[Nenonen, 2000] L. Nenonen, J. Gustafsson, J. Paakki A. Inkeri Verkamo, "Measuring object -oriented software architectures from UML diagrams", *Proc. 4th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, 2000, pp. 87-100.

[NASA,web] NASA's Definition of Risk Matrix, http://tkurtz.grc.nasa.gov/risk/rskdef.htm.

[NASA, 2000] NASA Safety Manual NPG 8715.3, Jan. 2000.

[NASA, 1997] NASA Technical Std. NASA-STD-8719.13A, *Software Safety*, 1997. http://satc.gsfc.nasa.gov/assure/nss8719_13.html

[NASA2, 1997] NASA Technical Std. NASA-STD-8719.13A, Software Safety, 1997.

[NASA3, 2000] *NASA Safety Manual NPG 8715.3*, Jan. 2000.

[Oman, 1997] P. Oman and S. L. Peeger, "Applying Software Metrics", IEEE Computer Society Press, Los Alamitos, CA, 1997.

[OMG, 2001] OMG Unified Modeling Language Specification Version 1.4, September 2001, http://www.uml.org.

[Papadopoulos, 1999] Yiannis Papadopoulos, John A. McDermid, "Hierarchically Performed Hazard Origin and Propagation Studies", *Proceedings of SAFECOMP '99, 18th Intl Conference on Computer Safety, Reliability and Security*, 1999, 139-152, Lecture Notes in Computer Science, 1698:, Springer Verlag, Toulouse France.

[Poels, 1999] G. Poels, "On the use of a Segmentally Additive Proximity Structure to Measure Object Class Life cycle Complexity", in: R. Dumke And A. Arban (Eds.), Software Measurement: Current Trends in Research and Practice, Deutscher Universitats Verlag, Wiesbaden, Germany, 1999, pp. 61-79.

[Purao, 2003] Sandeep Purao, Vijay Vaishnavi, "Product Metrics for Object-Oriented Systems", *ACM Computing Surveys*, Vol. 35, No. 2, June 2003, pp. 191–221.

[Porter, 1993] A. Porter, "Using Measurement-Driven Modeling to Provide Empirical Feedback to Software Developers", *Journal of Systems and Software*, vol. 20, pp. 237-243, 1993.

[Poter, 1990] A. Porter, R. Selby, "Evaluating Techniques for Generating Metric-Based Classification Trees", *Journal of Systems and Software*, vol. 12, pp. 209-218, 1990.

[Rose,2001] Rational Rose Real-Time. http://www.rational.com/products/rosert/index.jtmpl.

[Roger, 1997] Roger Pressman, "software Engineering-A practioner's Approach, 4th ed. McGraw - Hill, 1997.

[Rosenberg, 1998] Linda H. Rosenberg, "Applying and Interpreting Object Oriented Metrics", *10th Annual Software Technology Conference*, April 19 - 23, 1998, Salt Lake City, Utah.

[Rosenberg, 1999] L. Rosenberg, R. Stapko and A. Gallo "Risk-based Object Oriented Testing", *24th Annual NASA Software Engineering Workshop*, December 1, 1999.

[Rosenberg, 2000] Linda H. Rosenberg, Ruth Stapko and Albert Gallo**, "**Risk-based Object Oriented Testing**",** 13th International Software Internet Quality Week (QW2000) San Francisco, California USA 30 May 2000 - 2 June 2000.

[Sammarco, J. P. E., 2003] Sammarco, J. P. E., "A Normal Accident Theory-Based Complexity Assessment Methodology for Safety-Related Embedded Computer systems", Ph.D. Dissertation, West Virginia University, CSEE Dept, 2003.

[Selic, 1994] Selic, B., G. Gullekson, and P. Ward, "Real-Time Object Oriented Modeling", John Wiley & Sons, Inc. 1994.

[Sherer, 1988 Susan A. Sherer, "Methodology for the Assessment of Software Risk", Ph.D. Diss., Wharton School, University of Pennsylvania, 1988.

[Shyam, 1998] Shyam R. Chidamber, David P. Darcy, and Chris F. Kemerer, "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis", *IEEE Transactions on Software Engineering*, V 24, No 8, Aug. 1998.

[Schneidewind, 1994] Schneidewind, N. F., "Validating metrics for ensuring Space Shuttle Flight software quality", *Computer*, 50-57, Aug., 1994.

[Shepper, 2001] Martin Shepperd, Gada Kadodapp," Comparing Software Prediction Techniques Using Simulation" IEEE Transaction on Software Engineering (TSE), Vol. 27, No. 11, PP 1014-1022, November 2001.

[Taghi, 1997] Taghi M. Khoshgoftaar, K. Ganesan, Edward B. Allen, Fletcher D. Ross, Rama Munikoti, Nishith Goel, and Amit Nandi., "Predicting fault-prone modules with case-based reasoning", IEEE *Proceedings of the 8th International Symposium on Software Reliability Engineering, (ISSRE '97),* pages 27--35, Albuquerque, NM USA, November 1997.

[Toshihiro, 1999] Toshihiro Kamiya, Shinji Kusumoto, Katsuro Inoue**, "**Prediction of Fault-proneness at Early Phase in Object-Oriented Development", *2nd IEEE International Symposium on Object-Oriented Real-Time Distributed Computing***,** May 02- 05, 1999**,** Saint-Malo, France.

[Trivedi, 2002] K. S. Trivedi, "Probability and Statistics with Reliability", *Queuing and Computer Science Applications,* 2nd ed., John Wiley & Sons, 2002.

[Wang, 2003] T.Wang, A. Hassan, A. Guedem, W. Abdelmoez, K. Goseva-Popstojanova, H. Ammar," Architectural Level Risk Assessment Tool Based on UML Specifications", *the 25th Int'l Conference on Software Engineering ICSE.2003*

[Yacoub, 1999] S. Yacoub, B. Cukic, and H. Ammar, "Scenario-based Reliability Analysis of Component-Based Software", *Proc. 10th Int'l Symp. Software Reliability Eng*. (ISSRE'99), Boca Raton, Florida, 1999, pp. 22-31.

[Yacoub, 2002] S. Yacoub and H. Ammar, “A Methodology for Architectural-Level Reliability Risk Analysis,” *IEEE Trans. Software Eng*, vol. 28, no. 6, June 2002, pp. 529-547. (A short version of this paper appeared in Proc. 11th Int’l Symp. Software Reliability Eng., Oct. 2000).

[Yacoub, 1999z] S. Yacoub, H. Ammar, and T. Robinson, “Dynamic Metrics for Object Oriented Designs”, *Proc. 6th Int’l Symp. Software Metrics* (Metrics’99), Boca Raton, Florida, 1999, pp 50-61.

[Yacoub, 2000z] S. Yacoub, T. Robinson, and H. Ammar, “A Matrix-Based Approach to Measure Coupling in Object-Oriented Designs", *Journal of Object Oriented Programming*, vol. 13, no. 7, Nov. 2000, pp. 8-19.

[Zuse, 1998] Horst Zuse , “A Framework of Software Measurement”, Publisher: Walter de Gruyter, Berlin, Genthinerstr. 13, 10785 Berlin, 1998.

[Zuse, 1998] Horst Zuse, “A Framework o Software Measurement”, walter de Gryter, Berlin-New York 1998 ].

# *Appendix A*

## Pace Maker Case Study

The scenarios of the pacemaker are presented in this appendix.

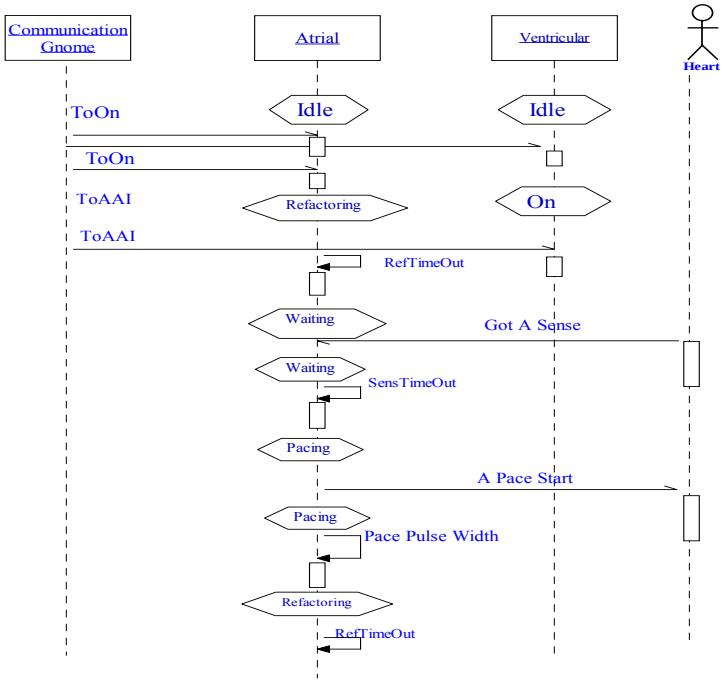



Figure 1 AAI scenario

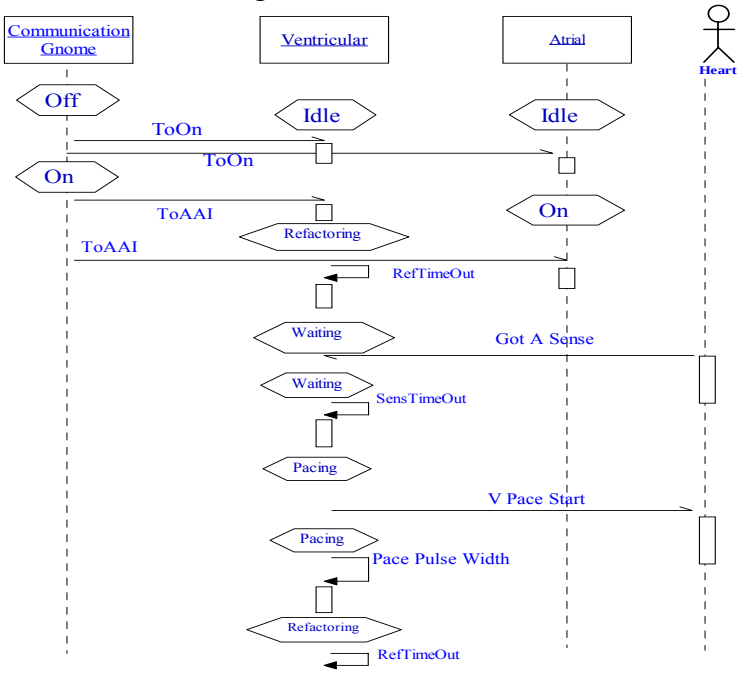



Figure 2 VVT Scenario

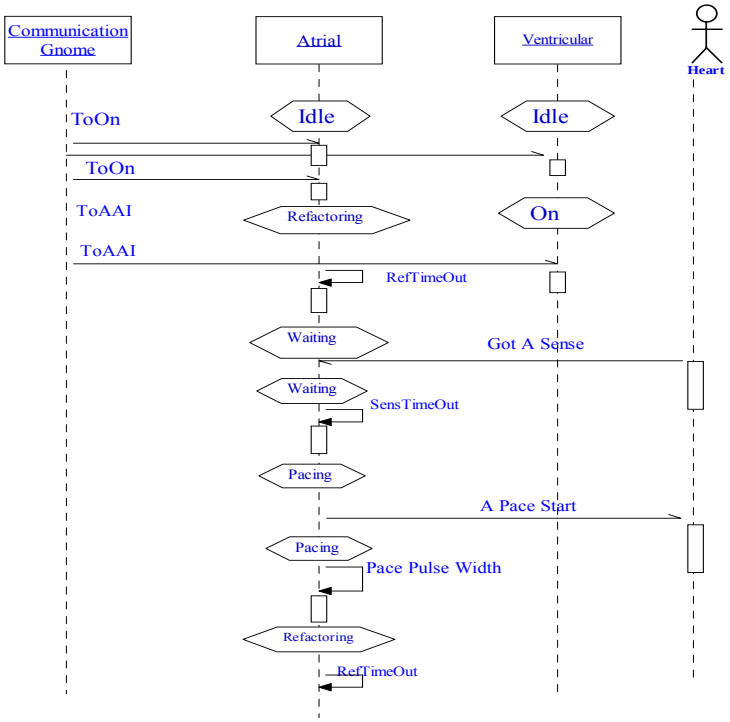



Figure 3 AAT Scenario

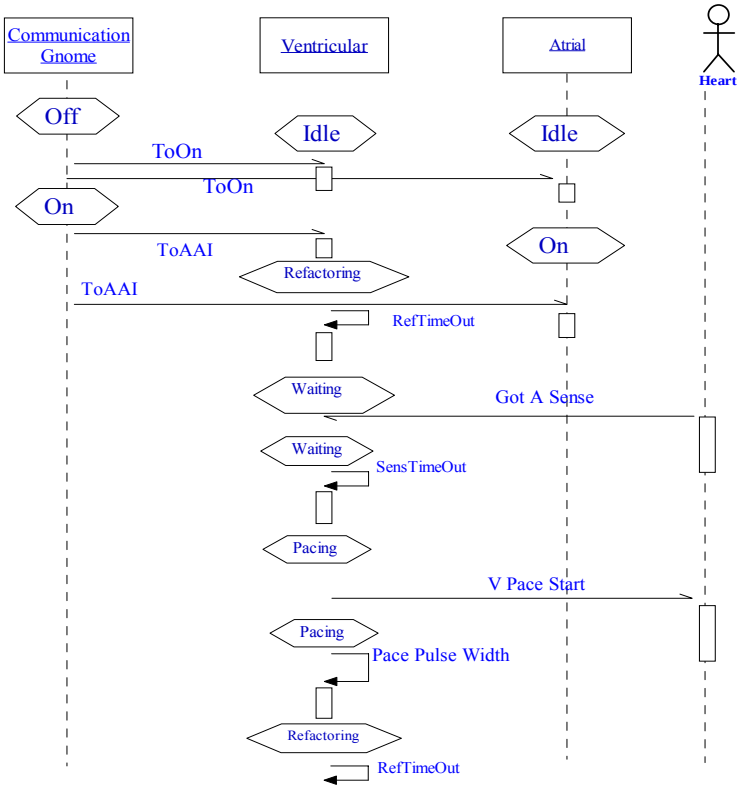



Figure 4 VVI scenario

Figures 1-4 show the scenario diagrams for the pace maker case study. Components and connectors severities are shown in Tables 1,2.

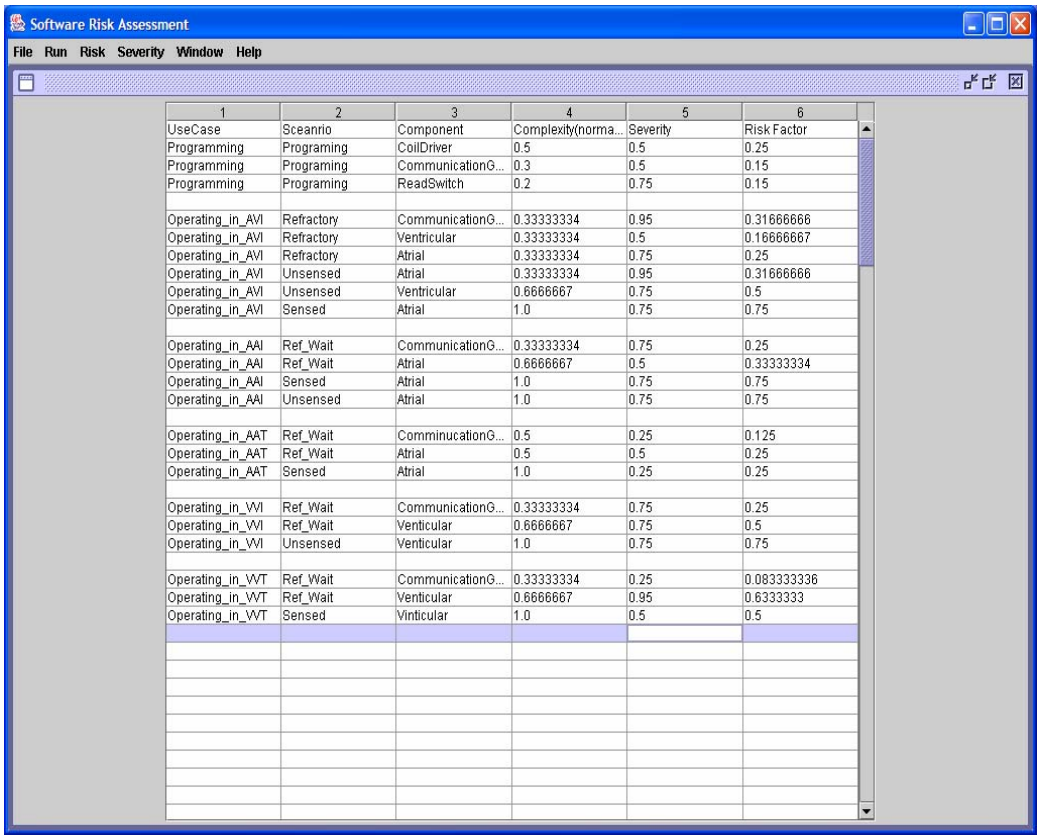
| Scenario Name | Component severity | | | | |
|---|---|---|---|---|---|
| | RS | CD | CG | AR | VT |
| *Programming* | 0.25 | 0.25 | 0.5 | | |
| *AVI* | | | 0.5 | 0.95 | 0.95 |
| *AAI* | | | 0.5 | 0.95 | |
| *VVI* | | | 0.5 | | 0.95 |
| *AAT* | | | 0.5 | 0.95 | |
| *VVT* | | | 0.5 | | 0.95 |

Table 1 components severity for every scenario of the Pace maker

| | | Scenario name | | | | | |
|---|---|---|---|---|---|---|---|
| | | Programming | AVI | AAI | VVI | AAT | VVT |
| Connector Name | RS-CD | 0.25 | | | | | |
| | RS-CG | 0.25 | | | | | |
| | CD-CG | 0.25 | | | | | |
| | CG-CD | 0.25 | | | | | |
| | CG-AR | | 0.5 | 0.5 | | 0.5 | |
| | CG-VT | | 0.5 | | 0.5 | | 0.5 |
| | VT-AR | | 0.95 | | | | |
| | AR-VT | | 0.95 | | | | |

Table 2 connectors severity for every scenario of the Pace maker

Figure 5 shows the risk factor for each component for each scenario of the pacemaker as shown from the tool.




| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| UseCase | Sceanrio | Component | Complexity(norma... | Severity | Risk Factor |
| Programming | Programing | CoilDriver | 0.5 | 0.5 | 0.25 |
| Programming | Programing | CommunicationG... | 0.3 | 0.5 | 0.15 |
| Programming | Programing | ReadSwitch | 0.2 | 0.75 | 0.15 |
| | | | | | |
| Operating_in_AVI | Refractory | CommunicationG... | 0.33333334 | 0.95 | 0.31666666 |
| Operating_in_AVI | Refractory | Ventricular | 0.33333334 | 0.5 | 0.16666667 |
| Operating_in_AVI | Refractory | Atrial | 0.33333334 | 0.75 | 0.25 |
| Operating_in_AVI | Unsensed | Atrial | 0.33333334 | 0.95 | 0.31666666 |
| Operating_in_AVI | Unsensed | Ventricular | 0.6666667 | 0.75 | 0.5 |
| Operating_in_AVI | Sensed | Atrial | 1.0 | 0.75 | 0.75 |
| | | | | | |
| Operating_in_AAI | Ref_Wait | CommunicationG... | 0.33333334 | 0.75 | 0.25 |
| Operating_in_AAI | Ref_Wait | Atrial | 0.6666667 | 0.5 | 0.33333334 |
| Operating_in_AAI | Sensed | Atrial | 1.0 | 0.75 | 0.75 |
| Operating_in_AAI | Unsensed | Atrial | 1.0 | 0.75 | 0.75 |
| | | | | | |
| Operating_in_AAT | Ref_Wait | CominucationG... | 0.5 | 0.25 | 0.125 |
| Operating_in_AAT | Ref_Wait | Atrial | 0.5 | 0.5 | 0.25 |
| Operating_in_AAT | Sensed | Atrial | 1.0 | 0.25 | 0.25 |
| | | | | | |
| Operating_in_VVI | Ref_Wait | CommunicationG... | 0.33333334 | 0.75 | 0.25 |
| Operating_in_VVI | Ref_Wait | Venticular | 0.6666667 | 0.75 | 0.5 |
| Operating_in_VVI | Unsensed | Venticular | 1.0 | 0.75 | 0.75 |
| | | | | | |
| Operating_in_VVT | Ref_Wait | CommunicationG... | 0.33333334 | 0.25 | 0.083333336 |
| Operating_in_VVT | Ref_Wait | Venticular | 0.6666667 | 0.95 | 0.6333333 |
| Operating_in_VVT | Sensed | Vinticular | 1.0 | 0.5 | 0.5 |

Figure 5 the tool demo output for risk of pace maker scenarios

Figure 6 shows the dynamic complexity of each component for every scenario of the pacemaker.

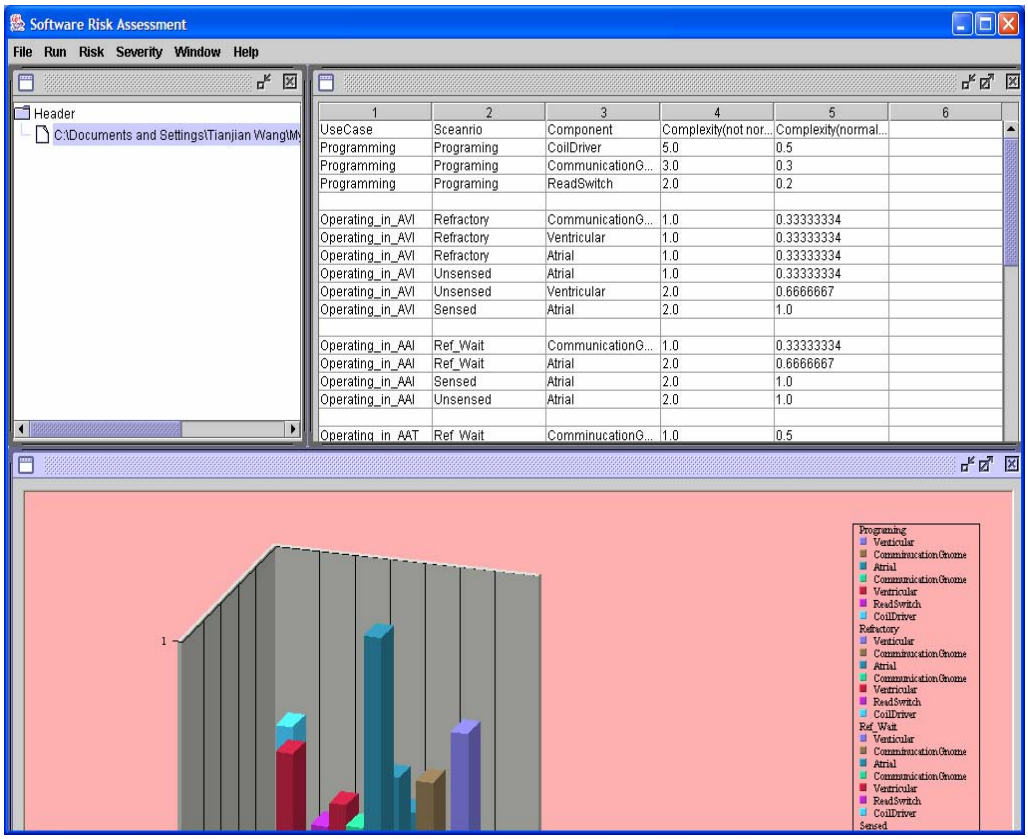



Figure 6 Dynamic Complexity of components of pace maker

# *Appendix B*

## HCS Case study Results

According to the expert domain knowledge the probabilities of use case operation is as follow.

| Use case | S_LT | D_LT | MT | S_MT | D_MT | R_B_P | Dua | LT | Monitor |
|---|---|---|---|---|---|---|---|---|---|
| Probability | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.92 |

Table (1) shows the probabilities of the scenarios of the HCS

Tables 2-9 show the complexity, severity and risk factor of every component for every scenario.

| scenario | component | complexity | severity | Risk Factor |
|---|---|---|---|---|
| S_LT | pFMC_MTR1 | 0.071428575 | 0.25 | 0.017857 |
| | pFMC_LTR1 | 0.14285715 | 0.75 | 0.107143 |
| | n3_2_Data_AccessR1 | 0.071428575 | 0.25 | 0.017857 |
| | appl_command_queueR1 | 0.071428575 | 0.75 | 0.053571 |
| | fRITCSR1 | 0.071428575 | 0.5 | 0.035714 |
| | sCITCSR1 | 0.37714287 | 0.95 | 0.358286 |
| | schedularR1 | 0.14285715 | 0.5 | 0.071429 |
| | n3_1_Data_AccessR1 | 0.071428575 | 0.75 | 0.053571 |

Table (2) S_LT Scenario components risk factor

| scenario | component | complexity | severity | Risk Factor |
|---|---|---|---|---|
| S_MT | pFMC_LTR1 | 0.071428575 | 0.25 | 0.017857 |
| | n3_1_Data_AccessR1 | 0.071428575 | 0.25 | 0.017857 |
| | pFMC_MTR1 | 0.14285715 | 0.75 | 0.107143 |
| | schedularR1 | 0.14285715 | 0.5 | 0.071429 |
| | sCITCSR1 | 0.35714287 | 0.95 | 0.339286 |
| | fRITCSR1 | 0.071428575 | 0.5 | 0.035714 |
| | n3_2_Data_AccessR1 | 0.071428575 | 0.75 | 0.053571 |
| | appl_command_queueR1 | 0.071428575 | 0.75 | 0.053571 |

Table (3) S_MT scenario components complexity, severity and risk factor

| scenario | component | complexity | severity | Risk Factor |
|---|---|---|---|---|
| R_B_P | sCITCSR1 | 0.2857143 | 0.95 | 0.27142859 |
| | fRITCSR1 | 0.35714287 | 0.95 | 0.33928573 |
| | rPCM_LT | 0.071428575 | 0.95 | 0.067857146 |
| | pFMC_LTR1 | 0.071428575 | 0.95 | 0.067857146 |
| | rPCM_MT | 0.14285715 | 0.95 | 0.13571429 |
| | pFMC_MTR1 | 0.071428575 | 0.95 | 0.067857146 |

Table (4) R_B_P Scenario components risk factor

| scenario | component | complexity | severity | Risk Factor |
|---|---|---|---|---|
| LT | n3_1_Data_AccessR1 | 0.18181819 | 0.95 | 0.172727 |
| | sCITCSR1 | 0.27272728 | 0.5 | 0.136364 |
| | pFMC_LTR1 | 0.09090909 | 0.75 | 0.068182 |
| | rPCM_LT | 0.18181819 | 0.5 | 0.090909 |
| | fRITCSR1 | 0.27272728 | 0.95 | 0.259091 |
| | appl_command_queueR1 | 0 | 0 | 0 |
| | schedularR1 | 0 | 0 | 0 |

Table (5) LT Scenario components risk factor

| scenario | component | complexity | severity | Risk Factor |
|---|---|---|---|---|
| D_LT | pFMC_MTR1 | 0.13333334 | 0.75 | 0.1 |
| | pFMC_LTR1 | 0.06666667 | 0.25 | 0.016667 |
| | fRITCSR1 | 0.06666667 | 0.5 | 0.033333 |
| | n3_2_Data_AccessR1 | 0.06666667 | 0.75 | 0.05 |
| | sCITCSR1 | 0.4 | 0.95 | 0.38 |
| | schedularR1 | 0.13333334 | 0.5 | 0.066667 |
| | n3_1_Data_AccessR1 | 0.06666667 | 0.25 | 0.016667 |
| | appl_command_queueR1 | 0.06666667 | 0.75 | 0.05 |

Table (6) D_LT Scenario components risk factor

| scenario | component | complexity | severity | Risk Factor |
|---|---|---|---|---|
| D_MT | n3_1_Data_AccessR1 | 0.06666667 | 0.75 | 0.05 |
| | appl_command_queueR1 | 0.06666667 | 0.75 | 0.05 |
| | fRITCSR1 | 0.06666667 | 0.5 | 0.033333 |
| | pFMC_MTR1 | 0.06666667 | 0.25 | 0.016667 |
| | sCITCSR1 | 0.4 | 0.95 | 0.38 |
| | schedularR1 | 0.13333333 | 0.5 | 0.066667 |
| | pFMC_LTR1 | 0.13333333 | 0.75 | 0.1 |
| | n3_2_Data_AccessR1 | 0.06666667 | 0.25 | 0.016667 |

Table (7) D_MT Scenario components risk factor

| scenario | component | complexity | severity | Risk Factor |
|---|---|---|---|---|
| MT | fRITCSR1 | 0.27272728 | 0.95 | 0.259091 |
| | sCITCSR1 | 0.27272728 | 0.5 | 0.136364 |
| | pFMC_MTR1 | 0.09090909 | 0.75 | 0.068182 |
| | n3_2_Data_AccessR1 | 0.18181819 | 0.95 | 0.172727 |
| | rPC_MT | 0.18181819 | 0.5 | 0.090909 |

Table (8) MT Scenario components risk factor

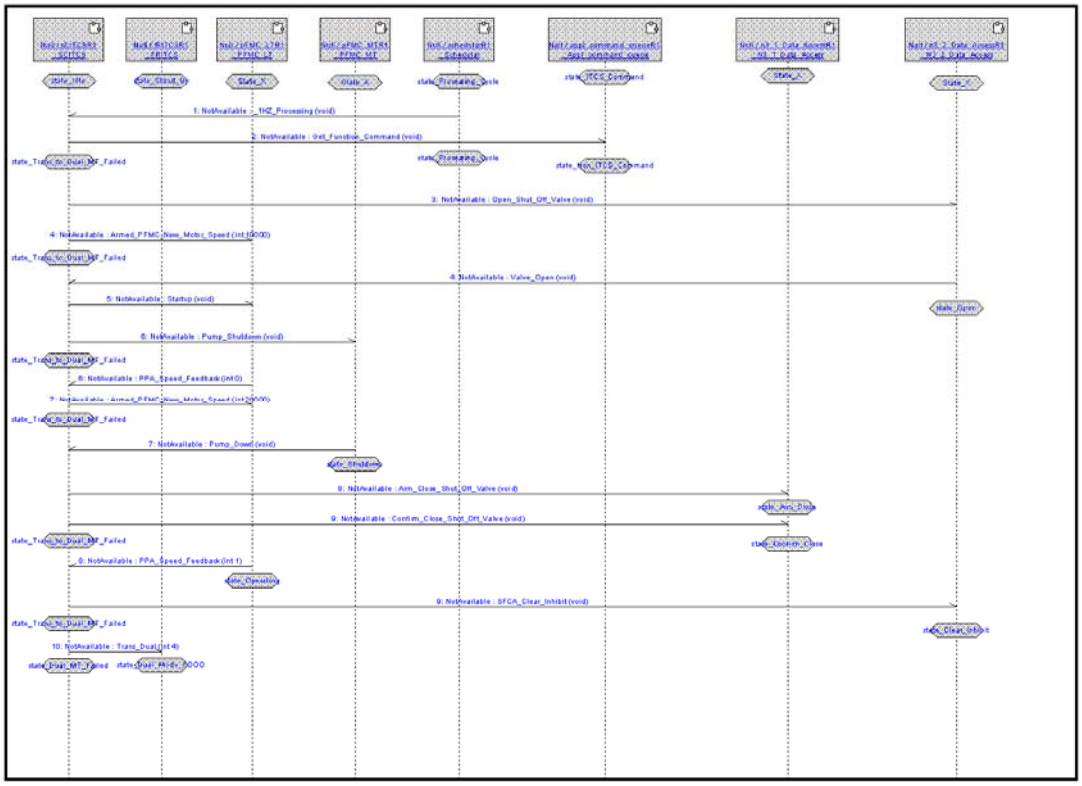


Figure 1 D_MT scenario

| scenario | component | complexity | severity | Risk Factor |
|---|---|---|---|---|
| Monitor | sCITCSR1 | 0.25 | 0.25 | 0.0625 |
| | appl_command_queueR1 | 0.25 | 0.95 | 0.2375 |
| | schedularR1 | 0.25 | 0.95 | 0.2375 |
| | fRITCSR1 | 0.25 | 0.95 | 0.2375 |

Table (9) Monitor scenario components risk factor

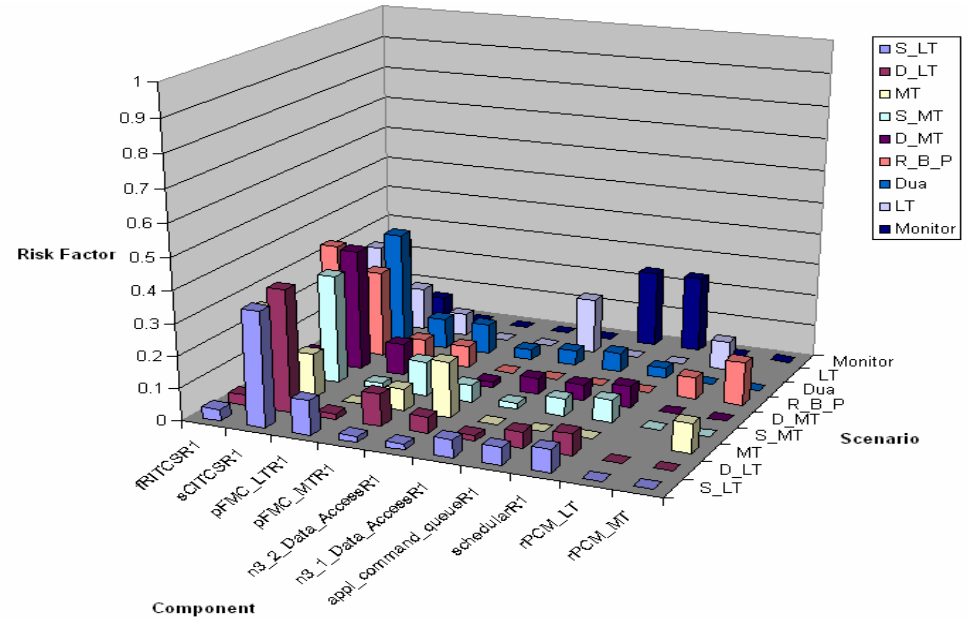



Figure 2 Components risk factor for every scenario

Figure 9 shows the 3-D bar graph for risk factor for all components for every scenario. The components “sCITCSR1” and “fRITCSR1” have high risk in more that one scenario.

**Connector risk factor for every scenario HCS**

Tables 10-18 show the connectors coupling, severity and risk factor for every scenari**o.**

| scenario | sender | receiver | coupling | severity | Risk Factor |
|---|---|---|---|---|---|
| S_MT | appl_command_queueR1 | sCITCSR1 | 0.05882353 | 0.5 | 0.029411765 |
| | sCITCSR1 | appl_command_queueR1 | 0.05882353 | 0.75 | 0.04411765 |
| | sCITCSR1 | pFMC_LTR1 | 0.05882353 | 0.95 | 0.055882353 |
| | sCITCSR1 | pFMC_MTR1 | 0.1764706 | 0.75 | 0.13235295 |
| | pFMC_LTR1 | sCITCSR1 | 0.05882353 | 0.95 | 0.055882353 |
| | sCITCSR1 | n3_2_Data_AccessR1 | 0.1764706 | 0.75 | 0.13235295 |
| | sCITCSR1 | fRITCSR1 | 0.05882353 | 0.75 | 0.04411765 |
| | sCITCSR1 | n3_1_Data_AccessR1 | 0.11764706 | 0.75 | 0.0882353 |
| | pFMC_MTR1 | sCITCSR1 | 0.11764706 | 0.75 | 0.0882353 |
| | n3_1_Data_AccessR1 | sCITCSR1 | 0.05882353 | 0.75 | 0.04411765 |
| | schedularR1 | sCITCSR1 | 0.05882353 | 0.95 | 0.055882353 |

Table (10) S_MT scenario

| scenario | sender | receiver | Coupling | severity | Risk Factor |
|---|---|---|---|---|---|
| S_LT | schedularR1 | sCITCSR1 | 0.05882353 | 0.75 | 0.04411765 |
| | sCITCSR1 | fRITCSR1 | 0.05882353 | 0.95 | 0.055882353 |
| | sCITCSR1 | appl_command_queueR1 | 0.05882353 | 0.5 | 0.029411765 |
| | n3_2_Data_AccessR1 | sCITCSR1 | 0.05882353 | 0.95 | 0.055882353 |
| | sCITCSR1 | pFMC_MTR1 | 0.05882353 | 0.95 | 0.055882353 |
| | sCITCSR1 | n3_1_Data_AccessR1 | 0.1764706 | 0.95 | 0.16764706 |
| | appl_command_queueR1 | sCITCSR1 | 0.05882353 | 0.75 | 0.04411765 |
| | sCITCSR1 | pFMC_LTR1 | 0.1764706 | 0.95 | 0.16764706 |
| | pFMC_LTR1 | sCITCSR1 | 0.11764706 | 0.75 | 0.0882353 |
| | sCITCSR1 | n3_2_Data_AccessR1 | 0.11764706 | 0.95 | 0.11176471 |
| | pFMC_MTR1 | sCITCSR1 | 0.05882353 | 0.75 | 0.04411765 |

Table (11) S_LT scenario

| scenario | sender | receiver | Coupling | severity | Risk Factor |
|---|---|---|---|---|---|
| Dua | sCITCSR1 | pFMC_MTR1 | 0.15 | 0.95 | 0.1425 |
| | sCITCSR1 | n3_2_Data_AccessR1 | 0.1 | 0.75 | 0.075 |
| | sCITCSR1 | n3_1_Data_AccessR1 | 0.1 | 0.75 | 0.075 |
| | n3_1_Data_AccessR1 | sCITCSR1 | 0.05 | 0.95 | 0.0475 |
| | pFMC_MTR1 | sCITCSR1 | 0.1 | 0.75 | 0.075 |
| | schedularR1 | sCITCSR1 | 0.05 | 0.75 | 0.0375 |
| | sCITCSR1 | pFMC_LTR1 | 0.15 | 0.75 | 0.112500004 |
| | sCITCSR1 | fRITCSR1 | 0.05 | 0.5 | 0.025 |
| | pFMC_LTR1 | sCITCSR1 | 0.1 | 0.95 | 0.095 |
| | sCITCSR1 | appl_command_queueR1 | 0.05 | 0.75 | 0.0375 |
| | n3_2_Data_AccessR1 | sCITCSR1 | 0.05 | 0.95 | 0.0475 |
| | appl_command_queueR1 | sCITCSR1 | 0.05 | 0.75 | 0.0375 |

Table (12) Dua scenario

| scenario | sender | receiver | Coupling | severity | Risk Factor |
|---|---|---|---|---|---|
| D_LT | schedularR1 | sCITCSR1 | 0.0625 | 0.75 | 0.046875 |
| | sCITCSR1 | fRITCSR1 | 0.0625 | 0.75 | 0.046875 |
| | sCITCSR1 | appl_command_queueR1 | 0.0625 | 0.75 | 0.046875 |
| | pFMC_MTR1 | sCITCSR1 | 0.125 | 0.75 | 0.09375 |
| | sCITCSR1 | pFMC_LTR1 | 0.0625 | 0.75 | 0.046875 |
| | sCITCSR1 | n3_2_Data_AccessR1 | 0.125 | 0.95 | 0.11875 |
| | sCITCSR1 | n3_1_Data_AccessR1 | 0.125 | 0.95 | 0.11875 |
| | appl_command_queueR1 | sCITCSR1 | 0.0625 | 0.95 | 0.059375 |
| | pFMC_LTR1 | sCITCSR1 | 0.0625 | 0.75 | 0.046875 |
| | sCITCSR1 | pFMC_MTR1 | 0.1875 | 0.75 | 0.140625 |
| | n3_1_Data_AccessR1 | sCITCSR1 | 0.0625 | 0.75 | 0.046875 |

Table (13) D_LT scenario

| scenario | sender | receiver | Coupling | severity | Risk Factor |
|---|---|---|---|---|---|
| Monitor | schedularR1 | sCITCSR1 | 0.33333334 | 0.75 | 0.25 |
| | sCITCSR1 | fRITCSR1 | 0.33333334 | 0.95 | 0.31666666 |
| | sCITCSR1 | appl_command_queueR1 | 0.33333334 | 0.75 | 0.25 |

Table (14) Monitor scenario

| scenario | sender | receiver | Coupling | severity | Risk Factor |
|---|---|---|---|---|---|
| LT | fRITCSR1 | sCITCSR1 | 0.3 | 0.95 | 0.285 |
| | n3_1_Data_AccessR1 | fRITCSR1 | 0.1 | 0.75 | 0.075 |
| | fRITCSR1 | pFMC_LTR1 | 0.1 | 0.75 | 0.075 |
| | pFMC_LTR1 | fRITCSR1 | 0.3 | 0.75 | 0.22500001 |
| | fRITCSR1 | rPCM_LT | 0.2 | 0.75 | 0.15 |

Table (15) LT scenario

| scenario | sender | receiver | Coupling | severity | Risk Factor |
|---|---|---|---|---|---|
| MT | fRITCSR1 | sCITCSR1 | 0.3 | 0.75 | 0.22500001 |
| | fRITCSR1 | pFMC_MTR1 | 0.1 | 0.75 | 0.075 |
| | n3_2_Data_AccessR1 | fRITCSR1 | 0.1 | 0.95 | 0.095 |
| | pFMC_MTR1 | fRITCSR1 | 0.3 | 0.95 | 0.285 |
| | fRITCSR1 | rPC_MT | 0.2 | 0.75 | 0.15 |

Table (16) MT scenario

| scenario | sender | receiver | Coupling | severity | Risk Factor |
|---|---|---|---|---|---|
| R_B_P | pFMC_LTR1 | fRITCSR1 | 0.2 | 0.5 | 0.1 |
| | fRITCSR1 | pFMC_LTR1 | 0.06666667 | 0.95 | 0.06333333 |
| | fRITCSR1 | pFMC_MTR1 | 0.06666667 | 0.75 | 0.050000004 |
| | pFMC_MTR1 | fRITCSR1 | 0.2 | 0.95 | 0.19 |
| | fRITCSR1 | sCITCSR1 | 0.26666668 | 0.75 | 0.20000002 |
| | fRITCSR1 | rPCM_LT | 0.06666667 | 0.75 | 0.050000004 |
| | fRITCSR1 | rPCM_MT | 0.13333334 | 0.75 | 0.10000001 |

Table (17) R_B_P scenario

| scenario | sender | receiver | Coupling | severity | Risk Factor |
|---|---|---|---|---|---|
| D_MT | sCITCSR1 | pFMC_LTR1 | 0.1875 | 0.75 | 0.140625 |
| | schedularR1 | sCITCSR1 | 0.0625 | 0.95 | 0.059375 |
| | sCITCSR1 | n3_1_Data_AccessR1 | 0.125 | 0.75 | 0.09375 |
| | appl_command_queueR1 | sCITCSR1 | 0.0625 | 0.95 | 0.059375 |
| | sCITCSR1 | pFMC_MTR1 | 0.0625 | 0.75 | 0.046875 |
| | n3_2_Data_AccessR1 | sCITCSR1 | 0.0625 | 0.75 | 0.046875 |
| | sCITCSR1 | appl_command_queueR1 | 0.0625 | 0.75 | 0.046875 |
| | sCITCSR1 | fRITCSR1 | 0.0625 | 0.75 | 0.046875 |
| | pFMC_LTR1 | sCITCSR1 | 0.125 | 0.75 | 0.09375 |
| | sCITCSR1 | n3_2_Data_AccessR1 | 0.125 | 0.75 | 0.09375 |
| | pFMC_MTR1 | sCITCSR1 | 0.0625 | 0.75 | 0.046875 |

Table (18) D_MT scenario

Applying the severity methodology presented in chapter 5 to this case study we estimate the severity of every component and connector for every scenario as shown in Table 19, Table 20.

| | | Scenario name | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | S_LT | D_LT | MT | LT | S_MT | D_MT | Dua | Monitor | R_B_P |
| Component name | pMR1 | 0.25 | 0.75 | 0.75 | 0 | 0.75 | 0.25 | 0.75 | 0 | 0.95 |
| | pLR1 | 0.75 | 0.25 | 0 | 0.75 | 00.25 | 0.75 | 0.75 | 0 | 0.95 |
| | n2R1 | 0.25 | 0.75 | 0.95 | 0 | 0.75 | 0.25 | 0.5 | 0 | 0 |
| | aR1 | 0.75 | 0.75 | 0 | 0 | 0.75 | 0.75 | 0.95 | 0.95 | 0 |
| | fR1 | 0.5 | 0.5 | 0.95 | 0.95 | 0.5 | 0.5 | 0.75 | 0.95 | 0.95 |
| | sR1 | 0.95 | 0.95 | 0.5 | 0.5 | 0.95 | 0.95 | 0.95 | 0.25 | 0.95 |
| | SchR1 | 0.5 | 0.5 | 0 | 0 | 0.5 | 0.5 | 0.25 | 0.95 | 0 |
| | n1R1 | 0.75 | 0.25 | 0 | 0.95 | 0.25 | 0.75 | 0.75 | 0 | 0 |
| | rLT | 0 | 0 | 0 | 0.5 | 0 | 0 | 0 | 0 | 0.95 |
| | rMT | 0 | 0 | 0.5 | 0 | 0 | 0 | 0 | 0 | 0.95 |

Table (19) Component severity for every scenario

| | | Scenario name | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | S_LT | D_LT | MT | LT | S_MT | D_MT | Dua | Monitor | R_B_P |
| Connector name | pMR1-sR1 | 0.75 | 0.75 | 0 | 0 | 0.75 | 0.75 | 0.75 | 0 | 0 |
| | pLR1-sR1 | 0.75 | 0.75 | 0 | 0 | 0.95 | 0.75 | 0.95 | 0 | 0 |
| | n2R1-sR1 | 0.95 | 0 | 0 | 0 | 0 | 0.75 | 0.95 | 0 | 0 |
| | aR1-sR1 | 0.75 | 0.95 | 0 | 0 | 0.5 | 0.95 | 0.75 | 0 | 0 |
| | fR1-sR1 | 0 | 0 | 0.75 | 0.95 | 0 | 0 | 0 | 0 | 0.75 |
| | sR1-aR1 | 0.5 | 0.75 | 0 | 0 | 0.75 | 0.75 | 0.75 | 0.75 | 0 |
| | SchR1-sR1 | 0.75 | 0.75 | 0 | 0 | 0.95 | 0.75 | 0.75 | 0.75 | 0 |
| | n1R1-sR1 | 0 | 0.75 | 0 | 0 | 0.75 | 0 | 0.95 | 0 | 0 |
| | rLT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | rMT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | sR1-pLR1 | 0.95 | 0.75 | 0 | 0 | 0.95 | 0.75 | 0.75 | 0 | 0 |
| | sR1-pMR1 | 0.95 | 0.75 | 0 | 0 | 0.75 | 0.75 | 0.95 | 0 | 0 |
| | sR1-n2R1 | 0.95 | 0.95 | 0 | 0 | 0.75 | 0.75 | 0.75 | 0 | 0 |
| | sR1-n1R1 | 0.95 | 0.95 | 0 | 0 | 0.75 | 0.75 | 0.75 | 0 | 0 |
| | sR1-fR1 | 0.95 | 0.75 | 0 | | 0.75 | 0.75 | 0.5 | 0.95 | 0 |
| | n1R1-fR1 | 0 | 0 | 0 | 0.75 | 0 | 0 | 0 | 0 | 0 |
| | fR1-pLR1 | 0 | 0 | 0 | 0.75 | 0 | 0 | 0 | 0 | 0.95 |
| | pLR1-fR1 | 0 | 0 | 0 | 0.75 | 0 | 0 | 0 | 0 | 0.5 |
| | fR1-rLT | 0 | 0 | 0 | 0.75 | 0 | 0 | 0 | 0 | 0.75 |
| | fR1-pMR1 | 0 | 0 | 0.75 | 0 | 0 | 0 | 0 | 0 | 0.75 |
| | pMR1-fR1 | 0 | 0 | 0.95 | 0 | 0 | 0 | 0 | 0 | 0.95 |
| | fR1-rMT | 0 | 0 | 0.75 | 0 | 0 | 0 | 0 | 0 | 0.75 |
| | n2R1-fR1 | 0 | 0 | 0.75 | 0 | 0 | 0 | 0 | 0 | 0 |

Table (20) Connector severity for every scenario